2003

The tuning factor effect in turbo coding

Tsen-Ying Lin
San Jose State University

THE TUNING FACTOR EFFECT IN TURBO CODING

A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

San Jose State University

In partial fulfillment

of the Requirements for the Degree

Master of Science

by

Tsen-Ying Lin

May 2003

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

APPROVED BY THE DEPARTMENT OF ELECTRICAL ENGINEERING

Dr. Essam Marouf

Dr. Avtar Singh

Dr. Robert H. Morelos Zaragoza

Dr. Masoud Mostafavi

APPROVED BY THE UNIVERSITY

ABSTRACT

THE TUNING FACTOR EFFECT IN TURBO CODING

by Tsen-Ying Lin

Channel coding is a technique used to combat channel impairments in digital and wireless communication systems, and turbo code is one of the latest channel coding technologies. This thesis presents a review of turbo encoder structures, decoding algorithms, and a discovery of a scaling coefficient of the extrinsic information in the iterative decoding process that dramatically changes the convergence speed and error performance of turbo codes. With the knowledge that this coefficient is tunable in nature, a new design of iterative turbo decoder based on this tuning factor and error detection is proposed to optimize the error performance. Simulations of single and multiple tuning factors decoding schemes are performed, and the results show that superior performance can be obtained when the tuning factor method is applied.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Essam Marouf for his constant support and precious advice to my work on the thesis. This thesis would not have been finished without his guidance, encouragement, and an endless shaping and review of my writing. I would also like to thank Dr. Robert H. Zaragoza and Dr. Avtar Singh for their great help and advice to solve the simulation problems I had encountered while the research was conducted.

In addition to the thanks listed above, I would like to extend special thanks to several people. First, thanks to my parents for their spiritual and financial support and giving me an opportunity to study abroad. Thanks also to my sisters Lisa and Faye, to my best friends Alex Hsu, Jay Song, Eric Yu, Choo Chin Tan, Justine Lai, Rich Huang, my cousins Jeffrey, Nancy, Morgan, and Sue, for their encouragement, caring about me, and being there for me. Finally, I would like to thank my girl, my love, Sandy Wang, for her understanding and company with me through out all the phases of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Digital communication has been widely involved in our daily life especially in the fields of mobile, satellite, and networking communications. In digital communication systems, the information is represented as a stream of binary bits during source coding processes. The binary bitstreams are then assigned (modulated) to analog signal waveforms and transmitted over a communication channel such as air, twisted pair telephone lines, and fiber optic cables. The communication channel introduces noise and interference that corrupts the transmitted signal and degrades the quality of service. At the receiver, the corrupted transmitted signal is mapped back to binary bits in the process of demodulation. The received binary information is an estimate of the transmitted binary information. Bit errors may occur during transmission, and the number of bit errors depends on how bad the transmitting environment is. Figure 1.1 shows is a general block diagram of a modern digital communication system.

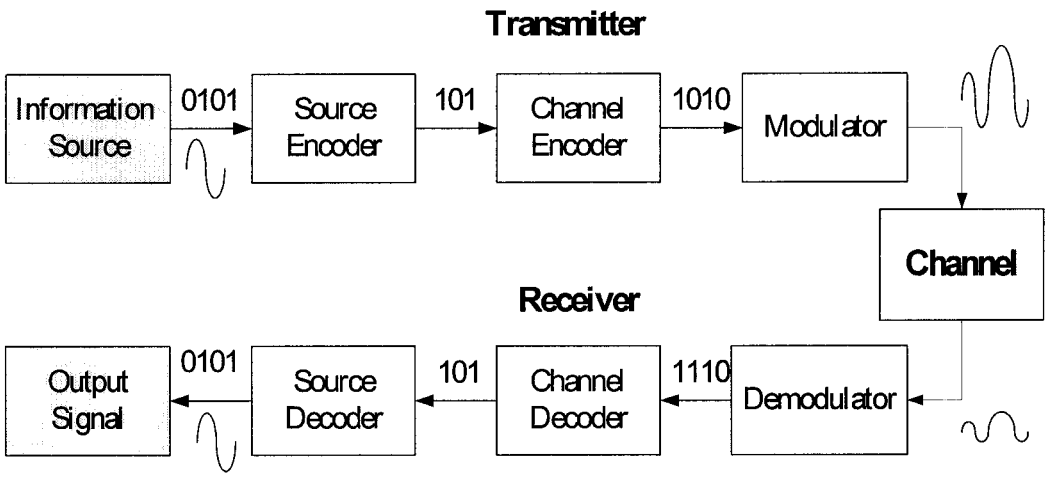


Figure 1.1: Block Diagram of Digital Communication System

For a relatively noisy channel (e.g., wireless communication channel), the probability of error may reach a value as high as 10^{-1}, which means only 9 out of 10 transmitted bits are received correctly. For many applications, this reliability is unacceptable. To achieve a high level of performance, for example, a bit error rate of 10^{-5}, channel coding was introduced.

Unlike source coding which removes the redundancy in the information sequence, channel coding adds protection or parity bits to the codeword. The decoder uses these parity bits in the decoding process at the receiver side and can increase the chance that the corrupted signal is recovered. In other words, the resistance of the digital communication systems to channel noise is enhanced. The cost of using channel coding to protect the information is a reduction in data rate, or a higher usage of bandwidth. But compared with the use of higher power transmitters and larger antennas, the cost of using channel coding is relatively low.

1.1 Types of Channel Codes

Channel coding can be partitioned into two study categories, waveform coding and structured sequences coding. Waveform coding deals with transforming waveforms into "better waveforms," which make the detection process less subject to errors [1]. "Structured sequence" coding deals with transforming data sequences into "better sequences", which contain a certain amount of redundancy bits.

In digital communications, it is the modulator's job to choose better waveforms that resist noise, interference, and fading. On the other hand, to find better sequences that protect the information bits and can better be recovered at the receiver side requires an appropriate encoding process and a corresponding decoding algorithm.

Because of their immediate relevance to the subject of this thesis, only structured sequences will be briefly discussed here. The two most known sequence structuring methods are block codes and convolutional codes.

1.1.1 Block Codes

Block codes are based rigorously on finite field arithmetic and linear algebra. They can be used to either detect or correct errors. Block codes accept a block of **k** information bits and produce a block of **n** coded bits. By predetermined rules, **(n-k)** redundant bits are added to the **k** information bits to form the **n** coded bits. Commonly, these codes are referred to as **(n, k)** block codes.

Generally, a block codes codeword can be represented as

$$\mathbf{C} = \mathbf{x} \cdot \mathbf{G} \tag{1.1}$$

where **x** is the input k bits information sequence, **G** is the code generator matrix, and **C** is the obtained codeword. The generator matrix **G** can be represented in a systematic form

$$\mathbf{G} = \left[\mathbf{P} \quad \mathbf{I}_k\right] \tag{1.2}$$

where **P** is a **k** by **(n-k)** matrix and $\mathbf{I}_k$ is the **k** by **k** identity matrix. The corresponding so called parity check matrix **H** is defined as

$$\mathbf{H} = \left[\mathbf{I}_{n-k} \quad \mathbf{P}^T\right] \tag{1.3}$$

where I_{n-k} is a **(n-k)** by **(n-k)** matrix and P^T is the transpose of **P**. The parity check matrix **H** is used in error detection to test whether a received vector is a codeword or not by checking

$$\mathbf{C} \cdot \mathbf{H}^T = 0 \tag{1.4}$$

Some of the well-known block codes are Hamming codes [2], Golay codes [3], BCH codes [4], and Reed Solomon codes (uses non-binary symbols) [5]. There are many ways to decode block codes and estimate the **k** information bits. Moreover, the codes can correct a certain number of bit errors in a corrupted received signal. Channel codes that have this ability are called *Error Correction Codes*.

In block codes, the encoder accepts a k-bit information block and generates an n-bit codeword. Intuitively, the code words are produced on a block-by-block basis. In digital communications, however, message bits come in serially rather than in large blocks. In this case, the use of a buffer is inevitable but undesirable. Another type of channel codes, called convolutional codes, can solve this problem.

1.1.2 Convolutional Codes

Convolutional codes were first introduced, as an alternative to block codes, by P. Elias in 1955 [6] and the ground work on algebraic theory was performed by G. D. Forney [7]. It is one of the most widely used channel codes in practical communication systems. Convolutional codes are developed with a separate strong mathematical structure and are primarily used for real time error correction. Convolutional codes convert the entire data stream into one single codeword bit-by-bit. Figure 1.2 shows a typical structure of a convolutional encoder with constraint length K=3, and rate=1/2, and u is the current input bit while v_1 and v_2 are two output code words.

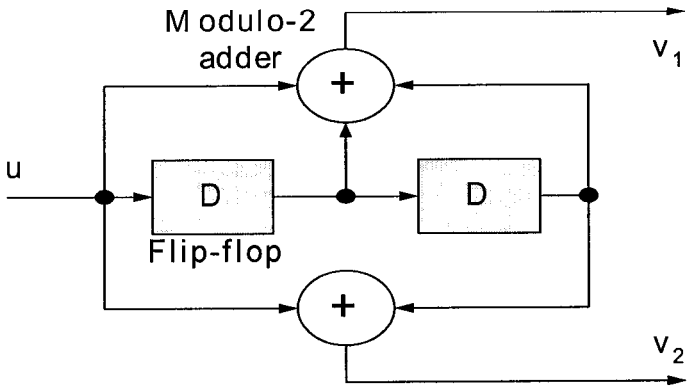


Figure 1.2: Convolutional Encoder with Constraint Length K=3, Rate=1/2

The encoder can be viewed as a finite-state machine consisting of an M-stage shift register with connections to a number of modulo-2 adders. The encoded bits depend not only on the current **k** input bits but also on past input bits that stored in the shift registers, which are the grey-colored flip-flops in Figure 1.2.

The main decoding strategy for convolutional codes is based on the widely known Viterbi algorithm [8], which finds the path of the shortest Hamming distance in the trellis diagram. The corresponding path represents the most likely codeword being transmitted. The greatest feature of convolutional codes is its simple structure, and the maximum likelihood soft decision decoding algorithm can be easily implemented. But with the consistent growth of demands for higher data rate, communication capacity, and better quality of service, better channel coding techniques and designs are imminently needed. Turbo codes, a new type of channel coding technique, seem to provide an alternative solution to this problem.

1.1.3 Turbo Codes

In 1993, a near-Shannon-Limit error correcting code called turbo code [9] was introduced. This error correcting code is able to transmit information across the channel with arbitrary low bit error rate. The code structure is a parallel

concatenation of two Recursive Systematic Convolutional (RSC) encoders. It has been shown that a turbo code can achieve performance within 0.7 dB of Shannon's channel capacity [10]. Without a doubt, the performance of a turbo code is partly due to the random interleaver used to scramble a burst of errors and provide randomness which improves the error performance during iterative decoding processes. Figure 1.3 shows the asymptotic error performance of turbo code [9]. Further discussions of turbo codes will be given in the following sections of this thesis.

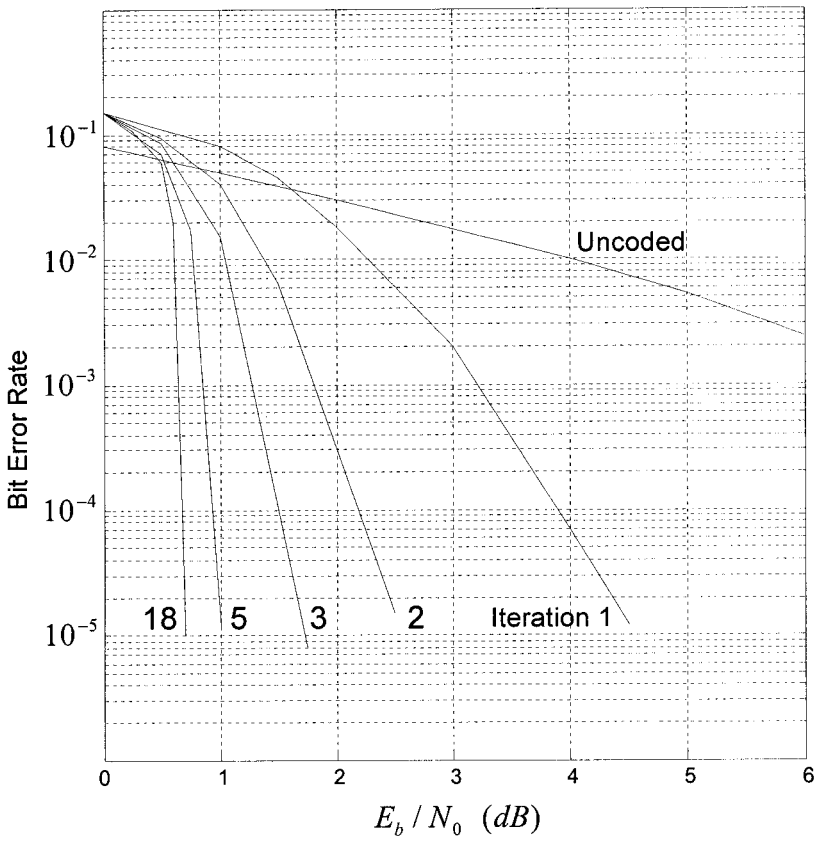


Figure 1.3: Error Performance of Turbo Code with 16-state, 65,536 bits, R=1/2

1.2 Outline of Thesis

The thesis is organized as follows. *Chapter 2* introduces the Recursive Systematic Convolutional (RSC) encoder used for turbo codes, and different ways to

construct turbo encoders. *Chapter 3* discusses the use and design of interleavers used for turbo codes, also the effect of interleaver size is examined. *Chapter* 4 presents different methods to decode the received data. *Chapter 5* introduces a new coding approach based on a novel tuning factor and examines its effects on the iterative decoding process. *Chapter 6* describes the numerical simulation setting used and the error pattern analysis conducted. Simulation results are presented. *Chapter 7* summarizes the main contribution of this thesis and discusses possible future work.

Chapter 2 Turbo Code Encoder

This chapter describes the fundamental Turbo code encoder structure and its components. The main encoder structure is the parallel concatenation of two identical recursive systematic convolutional (RSC) encoders [9]. The two RSC encoders are separated by a permuter (interleaver) which scrambles the information sequence at input of the second RSC encoder. The encoder output consists of three parts. The first is the input word **X** itself. And the other two are **Y1** and **Y2**, which are the output from RSC encoder 1 and encoder 2 respectively. The encoder is said to be systematic because the input bits are part of the codeword. Figure 2.1 shows the basic diagram of a turbo code encoder structure.

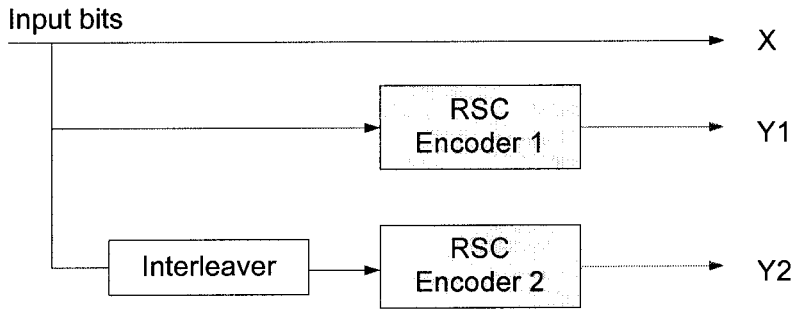


Figure 2.1: Turbo Code Encoder Structure

Notice that the permuted information sequence, that is the output of the interleaver, is discarded from the codeword. This is due to the fact that this systematic output is only a permuted version of the systematic bits **X**. As long as the receiver knows the relation (interleaving map) between the the input and output of the interleaver, the discarded sequence can be formed from the systematic bits **X** contained in the received signal. This also helps to increase the code rate.

2.1 Recursive Systematic Convolutional Encoder

The recursive systematic convolutional (RSC) encoder is obtained from the nonrecursive nonsystematic (conventional) convolutional encoder by feeding back one of its encoded outputs to its input. The RSC encoder can be represented by a set of code generator polynomial **G**=[1, g1/g0]. In this representation, "1" denotes the systematic bit, g0 is the encoder feedback polynomial, and g1 represents the feed-forward output. Figure 2.2 shows the resulting RSC encoder. And in this diagram, g0=[1 1 1 1] and g1=[1 0 0 1] respectively. The overall code generator polynomial **G** can be represented in octal as **G**=[1, 17/11].

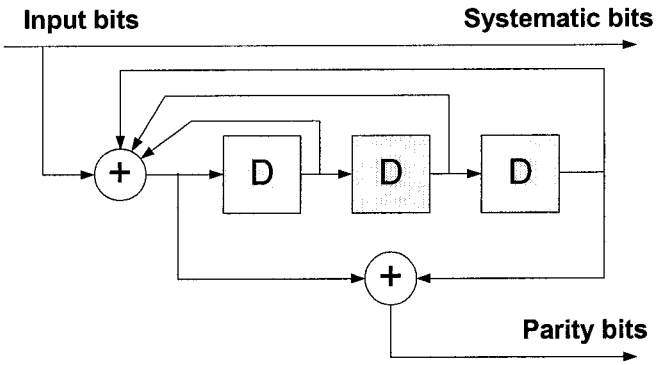


Figure 2.2: Recursive Systematic Convolutional Encoder

The code rate of this RSC encoder is r=1/2 with the fact that every input bit will correspond to two output bits – one is the systematic bit and another is parity bit. The memory order in Figure 2.2 is v=3 (the number of states is M=8). Typically, the most used RSC encoder in turbo code is of the memory order 2~4. Another way to represent the encoder is to use the generator matrix. In this example, the RSC encoder is given by

$$G(D) = \left[1, \quad \frac{1+D^3}{1+D+D^2+D^3}\right] \tag{2.1}$$

Let us assume the input sequence to be

$$\mathbf{X} = (1100101) \tag{2.2}$$

Then the first output sequence of the first component encoder is

$$\mathbf{Y1} = (1011000) \tag{2.3}$$

Suppose the interleaver permutes the information sequence into

$$\tilde{\mathbf{X}} = (1011001) \tag{2.4}$$

The parity check sequence of the second component is

$$\mathbf{Y2} = (1111100) \tag{2.5}$$

The turbo code sequence is given by

$$C = (111, 101, 011, 011, 101, 000, 100) \tag{2.6}$$

where each of the three grouped bits are from **X**, **Y1** and **Y2** respectively. And in this example, the code rate is r =1/3. To change the code rate, a technique called puncturing is used to discard one of the output bit from encoder 1 and 2 in a round-robin order. Thus for every input bit , there will be only two bits being generated by the encoder. The code rate is raised to r =1/2 consequently.

The use of the recursive systematic convolutional (RSC) encoder has two main reasons. First, the output of the codeword based on not only the current bit, but also previous input bits that are stored in the shift registers. It is this property that makes soft decision decoding possible. Second, the feedback of the output to the input makes the encoder generate a high code weight sequence even if the input sequence is a low weight sequence.

It is well known that at high per bit signal-to-noise ratio E_b/N_0 values, the error performance of a nonrecursive convolutional (NRC) code is better than that of a systematic code having the same memory order [11]. At small E_b/N_0 values, it is generally the other way around. And for high code rates, RSC codes result in better error performance than the best NSC code at any value of E_b/N_0 [12].

2.2 Concatenation of Encoders

There are two main types of concatenation for turbo codes. The first one is parallel concatenation, and this encoder architecture is proposed by Berrou and Glavieux in [9, 12, 13]. Another method of turbo code implementation is put the component encoders in serial concatenations [14, 15]. It has been suggested that serial concatenation of codes may have superior error performance compared with parallel concatenation [15].

2.2.1 Parallel Concatenated Convolutional Code (PCCC)

Figure 2.3 is an example of parallel concatenation with two component encoders. In general, there is no limit on the number of encoders that may be concatenated. And the component codes need not be identical with regard to constraint length and rate.

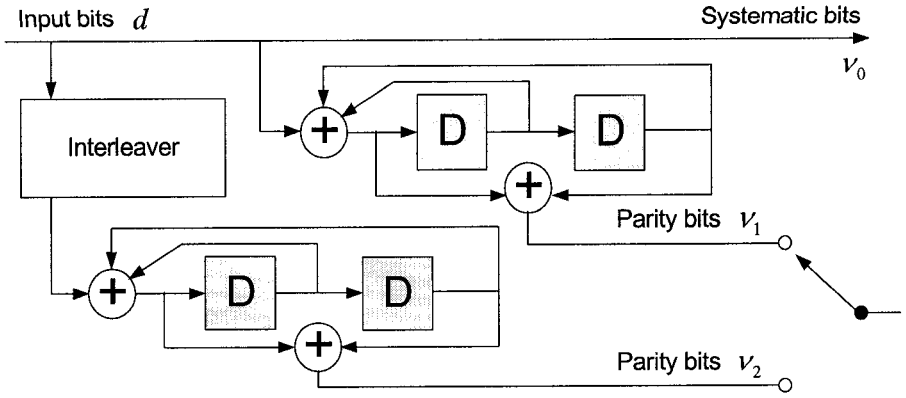


Figure 2.3: Parallel Concatenation of two RSC Encoders

A technique used to increase the code rate is puncturing. Instead of using all the parity bits in the output codeword, parity bits are selected alternatively. For example, if the parity bits v_1 and v_2 are selected every other codeword, then the overall code rate can be increased from $R = 1/3$ to $R = 1/2$. The puncturing table **P** in this case can be expressed as

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where the puncturing period is 2. However, the trade-off is the coding gain. Decoding complexity and speed are not usually affected by puncturing.

The state diagram and the trellis structure of the RSC encoder in Figure 2.3 are shown in Figure 2.4. Note that in the trellis diagram, the solid line represents the input 0, and the dashed line represents the input 1.

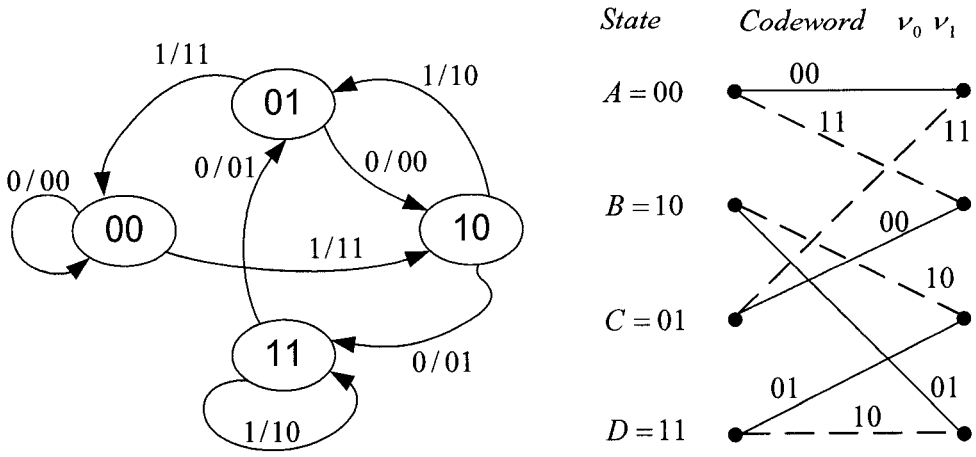


Figure 2.4: State and Trellis Diagram of the RSC Encoder with G=[1 5/7]

2.2.2 Serial Concatenated Convolutional Code (SCCC)

Another turbo encoder configuration is the serial concatenation. In the serial concatenation, an outer code C_o with rate $R_o = k/p$ and an inner code C_i with rate $R_i = p/n$ are connected by an interleaver of size N. The overall concatenated code rate is $R = R_o R_i = k/n$. Figure 2.5 shows the block diagram of the serial concatenation encoder.

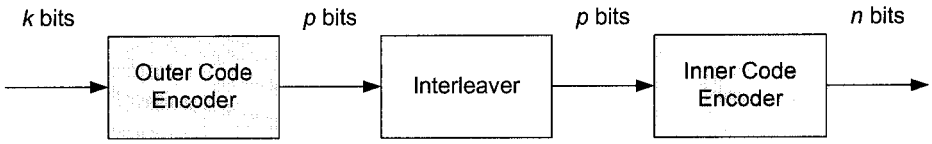


Figure 2.5: Block Diagram of Serial Concatenation

The type of turbo encoder adopted in this thesis is parallel concatenation, and the codeword generated by the corresponding finite-state machine is a Markov process that can be decoded by applying a Maximum a Posteriori (MAP) decoding algorithm, which will be introduced later in the thesis.

2.3 Low-weight Input Pattern

A rate $R = k/n$ turbo code with memory ν and interleaver length N can be represented as a $(n(N+\nu), kN)$ block code. The bit error probability of an equivalent block code decoded by a maximum likelihood algorithm over an additive white Gaussian noise channel can be upper bounded by a union bound [16]

$$P_b \le \sum_{d_{free}=d\min} B_d \cdot Q\left(\sqrt{d_{free} \cdot \frac{2RE_b}{N_0}}\right) \tag{2.7}$$

where R is the code rate, E_b / N_0 is the signal-to-noise ratio per information bit, d_{free} is the free distance of the codeword, $d_{\min}$ is the minimum Hamming distance, and B_d is the error coefficient. B_d is equal to the average number of bit errors caused by the transition between the all-zero codeword and codewords of weight $d_{free}(d_{free} \ge d_{\min})$. The $Q(\cdot)$ function is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-(u^2/2)} du \tag{2.8}$$

The set of (d_{free}, B_d) represents the turbo code distance spectrum which determines the contribution of the codewords with the same weight d_{free} to the bit error probability. A fast algorithm to calculate B_d is proposed in [17].

Note that the average upper bound is based on union bound techniques. It diverges when the ratio of information bit energy to noise power spectral density E_b / N_0 drops below the threshold determined by the computation cutoff rate R_0,

especially for large interleaver sizes. For a turbo code with rate $R = 1/3$, the threshold for E_b / N_0 is given by [18]

$$\frac{E_b}{N_0} = -\frac{1}{R}\ln\left(2^{1-R} - 1\right) = 2.03\ dB \tag{2.9}$$

For E_b / N_0 above the computation cutoff rate threshold, the average upper bound can be used to estimate the code performance; for E_b / N_0 below the threshold, the upper bound will be useless for accurate performance evaluations for turbo codes with large interleaver size. Several improved bounding techniques have been proposed to tighten the bounds at a E_b / N_0 below the cutoff rate threshold [19, 20, 21].

For a rate 1/3 turbo code with interleaver size 1024 bits, it has been shown [22] that at high SNR (2-8 dB), a small number of low weight spectral lines determine the code performance. And these codewords are generated by the input patterns with weight 2, 3, 4, 6, and 8. An example shows the influence of a weight 2 input pattern is presented below. In general, an RSC encoder can generate an infinite weight codeword while a convolutional encoder can not. Figure 2.6 compares the output sequence of an RSC and a convolutional encoder.

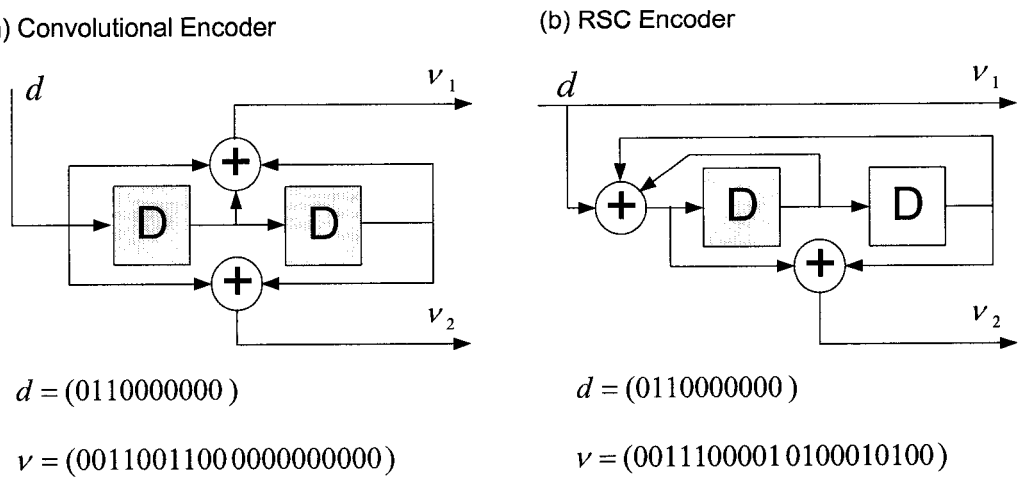


Figure 2.6: Output Sequence Comparison of Convolutional and RSC Encoders

It can be shown in Figure 2.6 that when a weight 2 input sequence is fed to a G=[1 1 1; 1 0 1] convolutional encoder, the weight of the output sequence is 4; with the same input sequence, the weight of the output sequence of a G=[1 5/7] RSC encoder is 7. The RSC encoder will generate an infinite weight output even if the input sequence is a low-weight infinite-length sequence.

Note that the distance of the input sequence d in the previous example is 1. That is, the two "1"s are separated by one position away from each other. In most cases, the RSC encoder can generate an output sequence of infinite weight. However, if the distance of the two "1"s are $K(v+1)$ distance apart, where v is the memory order of the encoder, $(v+1)$ is its constraint length, and K is a positive integer, then the output sequence will be of finite length.

Let us consider the RSC encoder in Figure 2.5. If the input sequence d' is

$$d' = (010010000000) \tag{2.10}$$

The two "1"s are 3 positions apart, then the corresponding output sequence v' is

$$v' = (001101011100000000000000) \tag{2.11}$$

The weight of the output sequence is 6 (which is of finite weight). Or if the input sequence d'' is

$$d'' = (0100000100000000) \tag{2.12}$$

The two "1"s are 6 positions apart, then the corresponding output sequence v'' is

$$v'' = (00110101000101110000000000000000) \tag{2.13}$$

The weight of the output sequence is 8 (of finite weight). Obviously, the free distance of these low weight codeword sequences is small, and this leads to a degradation in the error performance.

In other words, if the input of the turbo encoder has the property of the input pattern discussed above, we should prevent the input of the second RSC encoder to become the same low weight input pattern. We can achieve this goal with a proper designed interleaver that disallows this to happen. The design of the interleaver will be discussed in the next chapter.

2.4 Key Factors in Turbo Code Design

There are numbers of variables in the design and implementation of turbo codes that can greatly affect the overall performance and usability of the system. In general, these factors provide tradeoffs between coding gain, code rate, hardware complexity, decoding throughput, and decoding latency [23]. A brief discussion of these factors is given as below:

(1) *Block Length (Interleaver Size)*: The larger the block length is, the better the error performance will be. In other words, longer blocks provide better coding

gain (the original turbo code that came within 0.7 dB of the Shannon limit used a block length of 65,536 per frame). However, hardware complexity and decoding latency will scale up linearly with block length.

(2) *Constraint Length of RSC Encoder*: Increasing the memory order, that is, increasing the number of state of turbo encoders, will generally result in increasing the free distance d_{free} of the codeword. As Equation (2.7) shows, the free distance dominates the turbo code performance at high SNR's. Increasing the constraint length will improve the error performance. Generally, this is true only if the block length is also increased. Note that the decoding complexity increases exponentially with increasing constraint length.

(3) *Number of RSC Encoders*: Turbo codes typically use two RSC encoders. Although additional encoders can improve the error performance, the decoding complexity and latency increase immensely as each encoder requires a corresponding decoder and interleaver/deinterleaver pair in the decoder structure. Moreover, the code rate declines with the increase of RSC encoders unless heavy puncturing is used.

(4) *Puncturing and Number of Iterations*: The use of puncturing will directly increase the overall code rate at the expense of coding gain or more decoding iterations to get a better performance. In general, with more iterations in the decoding process, a better error performance can be achieved. However the improvement will diminish after the first few iterations, and each iteration will introduce additional latency.

The design of interleaver and the choice of decoding algorithm are another two important factors in turbo code design. Discussion about them will be presented in the following chapters.

Chapter 3 Interleaver Design

The interleaver structure and size play an important role in turbo code design. The choice of interleaver as well as its block size can considerably affect the error performance of turbo code. In this chapter, three types of interleaver will be presented. They are: block interleavers, random interleavers, and code-matched interleavers. First we discuss a block interleaver used in the original turbo code [9] where the interleaver spread out burst of errors and provided uncorrelated information between component decoders. Second, we show that with an appropriate design of interleaver, such as S-random interleaver [24], the weight of the output codeword and the corresponding free distance can be increased. An improved S-random permuter named code-matched interleaver will be presented, which eliminates the first few lines in the code distance spectrum and lowers the error floor at high SNR [22].

3.1 Interleaving in Turbo Codes

As disscussed in Chapter 2, the input of the second turbo encoder is a permuted version of the incoming sequence of the first encoder input. The interleaver is a device which rearranges the order of a data sequence in a one-to-one deterministic format. It is used in both the turbo encoder and decoder structure. The inverse of the interleaving process is deinterleaving which restores the received sequence to its original order. It is used in the turbo decoders.

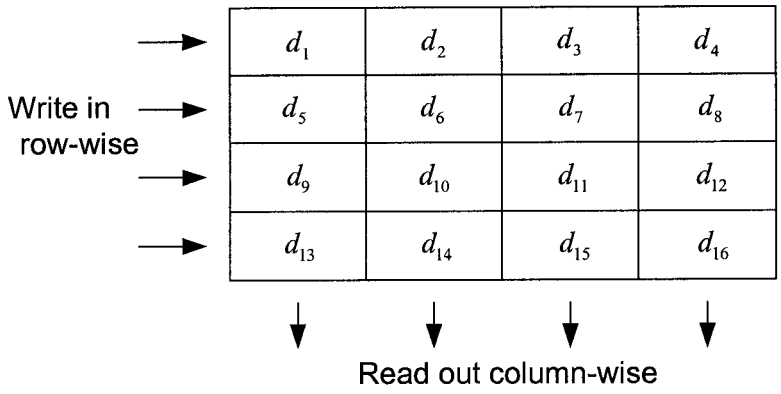
In turbo coding, the interleaver serves not only to spread out burst of errors, but also to provide scrambled information data to the second component encoder and hence decorrelates the inputs to the two component decoders. Thus, an iterative decoding algorithm based on uncorrelated information exchange between the two

component encoders can be applied. Another role of the interleaver is to break low weight input sequences which results in increasing the free Hamming distance of the codeword and hence improves the error performance.

3.2 Block Interleaver

A block interleaver is built-up in a matrix of m rows and n columns (or a block memory cell) which allows the input sequence to be written row-wise and read out column-wise. Due to this basic functionality it is also called a row-column interleaver. Table 3.1 is an example of how the data are written in and read out.

Table 3.1: Block Interleaver Example of Size 4×4



In this example, the size of the block interleaver is 4×4. The input sequence d is

$$d = [d_1\ d_2\ d_3\ d_4\ d_5\ d_6\ d_7\ d_8\ d_9\ d_{10}\ d_{11}\ d_{12}\ d_{13}\ d_{14}\ d_{15}\ d_{16}] \qquad (3.1)$$

and the corresponding output sequence $\tilde{d}$ of this interleaver is

$$\tilde{d} = [d_1\ d_5\ d_9\ d_{13}\ d_2\ d_6\ d_{10}\ d_{14}\ d_3\ d_7\ d_{11}\ d_{15}\ d_4\ d_8\ d_{12}\ d_{16}] \qquad (3.2)$$

Block interleanvers are easy to implement. However, they may fail to break

certain low weight input patterns or square input patterns. Table 3.2 presents an example in which the input and output sequences are the same. Note that in these patterns, the matrix is symmetric about the diagonal line.

Table 3.2: Block Interleaver Failure Example

1	0	0	0
0	0	1	1
0	1	0	0
0	1	0	0

The input and output sequences in this example are the same that

$$d = \tilde{d} = [1\,0\,0\,0\,0\,0\,1\,1\,0\,1\,0\,0\,0\,1\,0\,0] \tag{3.3}$$

In general, increasing the size of the block interleaver can reduce the chance of having an output sequence identical to the input sequence. There are several different interleaver designs which can prevent this kind of failure to happen. One of the most used design is a random interleaver.

3.3 Random Interleaver

A random interleaver is a variation of the block interleaver in which the data is written sequentially and read out in a random order [25]. A random interleaver can avoid a same-in-same-out even when the interleaver size is small. Two types of random interleaver are introduced in the following subsections.

3.3.1 Pseudorandom Interleaver

The pseudo-random interleavers are defined by a pseudo-random number

generator or a look-up table where all integers from 1 to N can be generated where N is the block size. Let $A = [1, 2, \Lambda, N]$ denotes the pool from which the interleaver output sequence $\pi(i)$ $(i \in 1, 2, \Lambda, N)$ will be randomly chosen. And whenever an integer is assigned to $\pi(i)$, remove that integer from the pool A so that it won't be chosen again. Repeating this step for N times until every number in set A is corresponding to an interleaver output, the interleaving map π is obtained. Note that the probability for an integer in set A to be chosen is equally likely that $p(A(i)) = 1/N$.

3.3.2 S-random Interleaver

The S-random interleavers proposed by Divsalar and Pollara [24] are pseudo-random interleavers. The interleaver output are produced on the random generation of N integers with an S-constraint, where S is defined as the minimum interleaving distance.

The selection of the interleaver output is based on the following rule: each randomly selected integer is compared to S previously selected integers. If the current integer is at a distance larger than S from any of the S previous selections, then the current selected integer can be the interleaver output; otherwise it will be discarded. The process is repeated till all N integers are selected. For a given N, the maximum S constraint is set to be less than $\sqrt{N/2}$, and the interleaver gain is usually large for larger S values.

For a turbo encoder, an S-random interleaver can break the input patterns with lengths up to $(S+1)$ and generate high weight parity check sequences. Or it can expand low weight input patterns to longer error patterns with length more than $(w-1)(S+1)$, where w is the input sequence weight. Therefore S-random

interleavers can achieve better performance compared to pseudo-random interleavers.

In this thesis, both pseudo-random and *S*-random interleavers will be investigated using simulations. A comparison of these two types will be given in Chapter 6.

3.4 Code-matched Interleaver

A code-matched interleaver proposed by Vucetic et al [22] is constructed to match the code weight distribution of turbo codes. With a code-matched interleaver, the low weight paths in the code trellis which give large contributions to the error probability in the high SNR region are eliminated. Hence the error performance is improved and the error floor can be lowered.

Recall the low weight input patterns discussed in Section 2.3, Figure 3.1 graphically shows the formation of a weight 2 input pattern where P and Q are the input of the input of the first and second encoders.

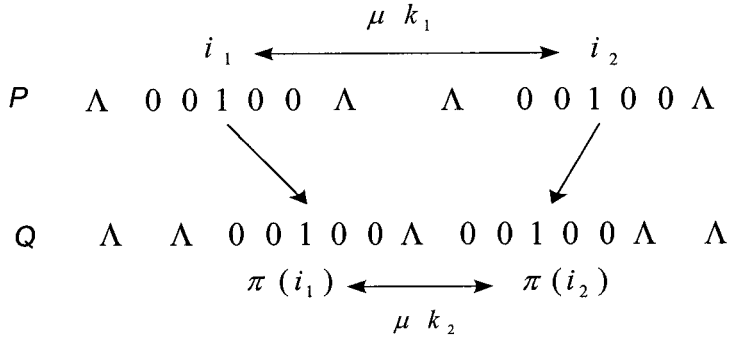


Figure 3.1: Low Weight Input Pattern of Weight 2

Note that Q is the interleaved version of P, μ is the constraint length of the RSC encoder, $i_1, i_2, \pi(i_1)$ and $\pi(i_2)$ are the positions of "1"s of the input sequences P and Q, and k_1 and k_2 are positive integers. If an interleaver mapping function meets the following condition

$$| i_1 - i_2 | \bmod \mu = 0 \qquad \textit{and} \qquad | \pi(i_1) - \pi(i_2) | \bmod \mu = 0 \qquad (3.4)$$

this interleaver will map the input sequence into another weight-2 input sequence that generates a finite weight parity check sequence as illustrated in Figure 3.1. In order to avoid this type of mapping, the mapping condition set ϕ should satisfy the following conditions:

$$| \pi(i_1) - \pi(i_2) | \bmod \mu \neq 0 \quad \textit{whenever} \quad | i_1 - i_2 | \bmod \mu = 0 \qquad (3.5)$$

A similar weight-4 input pattern that generates a finite weight parity check sequence is depicted in Figure 3.2.

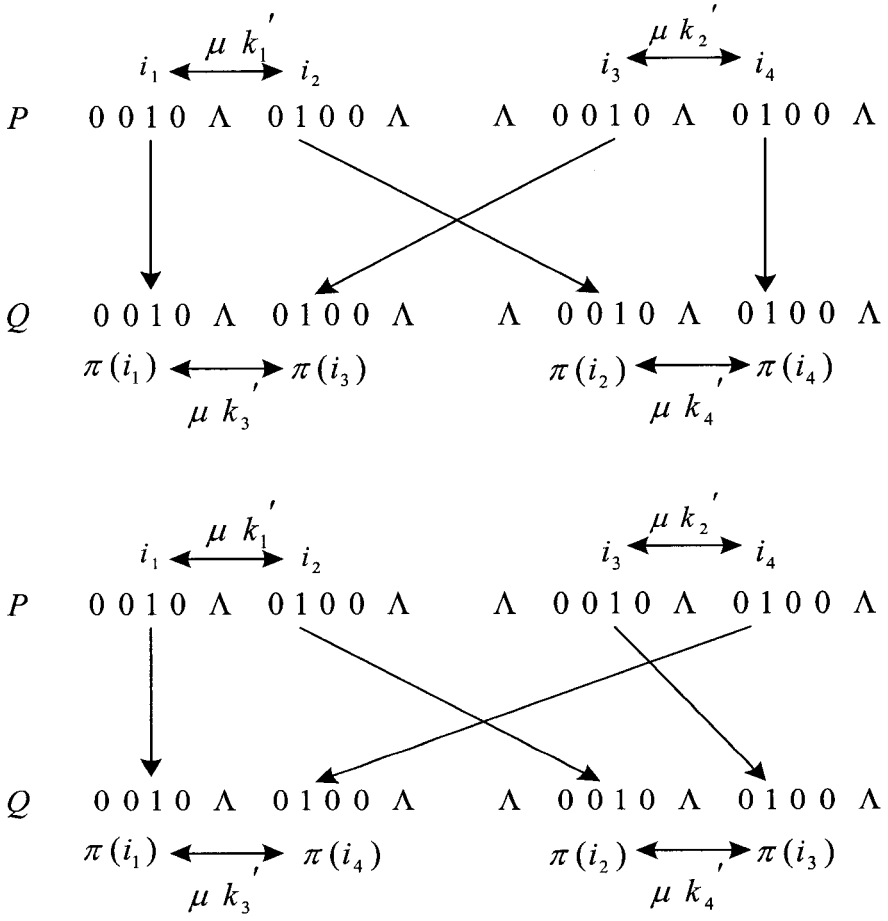


Figure 3.2: Two Examples of Low Weight Input Pattern of Weight 4

where i_1, i_2, i_3 and i_4 denote the positions of "1"s in the weight-4 input sequence P, i_1, i_2, i_3 and $i_4 \in A$ and $i_2 < i_2 < i_3 < i_4$; similarly, $\pi(i_1)$, $\pi(i_2)$, $\pi(i_3)$ and $\pi(i_4)$ denote the positions of "1"s in the interleaved sequence Q, and k_1', k_2', k_3' and k_4' are positive integers. If an interleaver mapping function meets the following conditions

$$\begin{cases} |i_1 - i_2| \bmod \mu = 0 \quad and \quad |i_3 - i_4| \bmod \mu = 0 \\ |\pi(i_1) - \pi(i_3)| \bmod \mu = 0 \\ and\ |\pi(i_2) - \pi(i_4)| \bmod \mu = 0 \end{cases} \tag{3.6}$$

or

$$\begin{cases} |i_1 - i_2| \bmod \mu = 0 \quad and \quad |i_3 - i_4| \bmod \mu = 0 \\ |\pi(i_1) - \pi(i_4)| \bmod \mu = 0 \\ and\ |\pi(i_2) - \pi(i_3)| \bmod \mu = 0 \end{cases} \tag{3.7}$$

then this interleaver will map the input sequence to another weight-4 sequence that includes two weight-2 input patterns. In order to avoid this type of mapping, the following condition should be added to ϕ:

$$\begin{cases} \quad |\pi(i_1) - \pi(i_3)| \bmod \mu \neq 0 \\ and \quad |\pi(i_2) - \pi(i_4)| \bmod \mu \neq 0 \\ whenever \\ \quad |i_1 - i_2| \bmod \mu = 0 \\ and \quad |i_3 - i_4| \bmod \mu = 0 \end{cases} \tag{3.8}$$

or

$$\begin{cases} \quad |\pi(i_1) - \pi(i_4)| \bmod \mu \neq 0 \\ and \quad |\pi(i_2) - \pi(i_3)| \bmod \mu \neq 0 \\ whenever \\ \quad |i_1 - i_2| \bmod \mu = 0 \\ and \quad |i_3 - i_4| \bmod \mu = 0 \end{cases} \tag{3.9}$$

Combined with the S-constraint and the mapping condition set ϕ, a

code-matched interleaver can be constructed. By eliminating these low weight input patterns and with a higher free distance in the codeword, an improvement in error performance is promising. The error floor predicted by the upper bound in Equation (2.7) in the high SNR region can now be lowered as well.

Chapter 4 Iterative Turbo Coder Decoder

In Chapter 2 and 3, the main components of the turbo encoder structure have been introduced. In this Chapter, various turbo decoding algorithms will be examined and compared. The first method is called maximum a posteriori (MAP) probability decoding which provides optimum reliability estimation of the received signal. Then various versions of the MAP algorithms will be shown, such as Log-MAP, which provides a trade-off between performance and complexity. Another alternative suboptimum decoding method is the soft output Viterbi algorithm (SOVA). The difference between SOVA and Viterbi algorithm is that SOVA returns a real number soft output, where the Viterbi algorithm makes the hard decision of the received bit sequence.

Although the turbo code is by far one of the best performance channel coding techniques, this outstanding performance is at the expense of burdensome computations and high hardware complexity. To apply turbo codes in real-time communication systems, some criteria are set to stop the iterative decoding process if little improvement is achieved from further computation. This technique can provide a significant increase in the average decoding speed without sacrificing decoder performance. Discussion about stopping rules will be presented in the last section of this chapter.

4.1 MAP Algorithm

The component turbo decoders require soft inputs and produce soft outputs for the iterative decoding process. Special decoding algorithms such as the maximum a

posteriori (MAP) [26] and the Log-MAP [27] algorithms can be invoked, which were proposed by Bahl and Robertson, respectively. Essentially, the soft output generated by either decoder determines whether the decoded bit is a binary 1 or 0 as well as the reliability of the output bit decision.

4.1.1 General Definition

The MAP algorithm minimizes the bit (or symbol) error probability. For each transmitted symbol it generates its hard estimate and soft output in the form of the a posteriori probability on the basis of the received sequence **r**. The mathematical foundation rests on Bayes' theorem. For communications engineering, where applications involving an AWGN channel are of great interest, a useful form of Bayes' theorem expresses the a posteriori probability (APP) of a decision in terms of a continuous-valued random variable **r** as

$$\mathbf{P}(d = i \mid r) = \frac{p(r \mid d = i)\mathbf{P}(d = i)}{p(r)} \qquad i = 1, 2, ..., M \tag{4.1}$$

and

$$p(r) = \sum_{i=1}^{M} p(r \mid d = i)\mathbf{P}(d = i) \tag{4.2}$$

where **P**(d= i | r) is the APP, and **d** = i represents data **d** belonging to the i-th signal class from a set of M classes. Further, $p(r \mid d = i)$ represents the probability density function (pdf) of a received continuous-valued and noise-corrupted signal **r**, conditioned on the signal class d = i. Also, p(d = i), called a priori probability, is the probability of occurrence of the signal i. Typically **r** is an "observable" random variable or a test statistic that is obtained at the output of a demodulator or some other

signal processor.

Let the binary logical elements 1 and 0 be represented by voltages +1 and –1, respectively. The variable **"d"** is used to represent the transmitted data bit, whether it appears as a voltage or as a logical element. For signal transmission over an AWGN channel, Figure 4.1 shows the conditional pdfs, referred to as likelihood functions. The rightmost function P(r | d = +1) shows the pdf of the random variable r conditioned on d= +1 being transmitted. And the leftmost function P(r | d = -1) illustrates a similar pdf conditioned on d = -1 being transmitted. The horizontal axis represents the full range of possible values of the test statistic **r** generated at the receiver. In Figure 4.1, one such arbitrary value R_k is shown, where the index k denotes an observation in the k-th time interval. A vertical line at R_k intercepts the two likelihood functions yielding two likelihood values $\lambda_1 = P(r_k \mid d_k = +1)$ and $\lambda_2 = P(r_k \mid d_k = -1)$. A well-known hard-decision rule, known as maximum likelihood (ML), is to choose the data $d_k = +1$ or $d_k = -1$ associated with the larger of the two intercept values λ_1 and λ_2, respectively. For each data bit at time k, this is tantamount to deciding that $d_k = +1$ if R_k falls on the right side of the center line r = 0, otherwise deciding that $d_k = -1$.

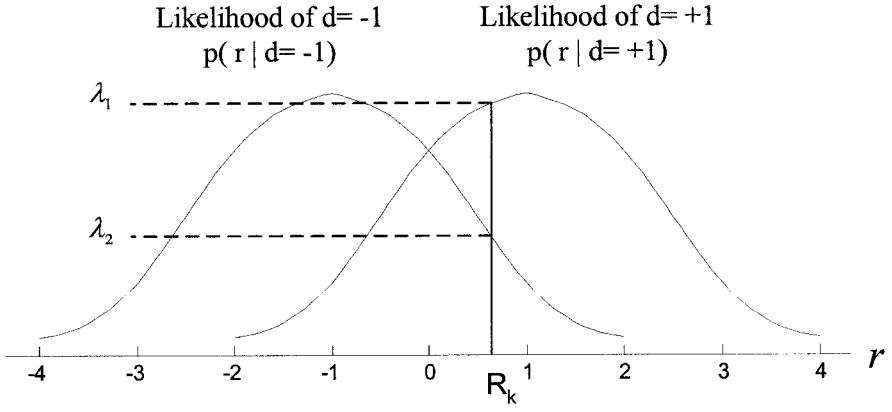


Figure 4.1: Likelihood Functions

A similar decision rule, known as maximum a posteriori (MAP), which can be shown to be a minimum-probability-of-error rule, takes into account the a priori probabilities of the data. The general expression for the MAP rule is

$$P(\mathrm{d}=+1\,|\,\mathrm{r}) \underset{H2}{\overset{H1}{\gtrless}} P(\mathrm{d}=-1\,|\,\mathrm{r}) \tag{4.3}$$

Equation (4.3) states that one should choose hypothesis H1, which is d = +1, if the APP P(d = +1| r), is greater than the APP P(d = -1| r). Otherwise, one should choose hypothesis H2, which is d = -1. Using Bayes' theorem

$$p(r \mid d=+1)P(d=+1) \underset{H2}{\overset{H1}{\gtrless}} p(r \mid d=-1)P(d=-1) \tag{4.4}$$

Equation (4.3) is generally expressed in terms of a ratio, yielding the so-called likelihood ratio as follows:

$$\frac{p(r \mid d=+1)}{p(r \mid d=-1)} \underset{H2}{\overset{H1}{\gtrless}} \frac{P(d=-1)}{P(d=+1)} \tag{4.5}$$

or

$$\frac{p(r \mid d=+1)}{p(r \mid d=-1)} \frac{P(d=+1)}{P(d=-1)} \underset{H2}{\overset{H1}{\gtrless}} 1 \tag{4.6}$$

Taking the logarithm of the likelihood ratio in Equations (4.3) through (4.6), a useful metric called the log-likelihood ratio (LLR) can be obtained. The LLR is a real number representing a soft decision out of a detector, designated by

$$L(d \mid r) = \log\left[\frac{P(d=+1 \mid r)}{P(d=-1 \mid r)}\right] \tag{4.7}$$

where by Bayes' rule

$$P(d,r) = p(d \mid r)P(r) = p(r \mid d)P(d)$$

and P(r) is a scaling factor obtained by averaging over all signals in the space. Thus

$$P(d \mid r) = p(r \mid d)P(d)$$

We can rewrite equation (4.7) into

$$L(d \mid r) = \log\left[\frac{p(r \mid d=+1)P(d=+1)}{p(r \mid d=-1)P(d=-1)}\right] \tag{4.8}$$

$$L(d \mid r) = \log\left[\frac{p(r \mid d=+1)}{p(r \mid d=-1)}\right] + \log\left[\frac{P(d=+1)}{P(d=-1)}\right] \tag{4.9}$$

or

$$L(d \mid r) = L(r \mid d) + L(d) \tag{4.10}$$

where $L(r \mid d)$ is the log-likelihood ratio (LLR) of the test statistics obtained by measurements of the channel output **r** under the alternate conditions that d = +1 or d = -1 may have been transmitted. $L(r \mid d)$ is the a posteriori probability and $L(d)$ is the a priori LLR of the data bit. Equation (4.10) can be rewritten as

$$L'(\hat{d}) = L_c(r) + L(d) \tag{4.11}$$

where the notation $L_c(r)$ denotes that this LLR term is the result of a channel measurement made at the receiver. For a systematic code, it can be shown [9] that the LLR (soft output) of the decoder is equal to

$$L(\hat{d}) = L'(\hat{d}) + L_e(\hat{d}) \tag{4.12}$$

where $L'(\hat{d})$ is the LLR of a data bit out of the demodulator (input to the decoder), and $L_e(\hat{d})$, called the extrinsic LLR, represents extra knowledge that is gleaned from the decoding process. The output sequence of a systematic decoder is made up of values representing data bits and parity bits. From Equations (4.11) and (4.12), the output LLR of the decoder is now written as

$$L(\hat{d}) = L_c(r) + L(d) + L_e(\hat{d}) \tag{4.13}$$

Equation (4.13) shows that the output LLR of a systematic decoder can be represented as having three LLR elements (i) a channel measurement, (ii) a priori knowledge of the data, and (iii) an extrinsic LLR stemming from the decoder. To yield the final $L(\hat{d})$, the individual LLRs can be added as given in Equation (4.13), because these three terms are statistically independent [9, 28].

4.1.2 BCJR Algorithm

The MAP based algorithm adopted by the original turbo code is a modified BCJR algorithm [9, 29, 30], which generates the a posteriori probability (APP) soft output for each decoded bit. This soft output is to be used in the iterative turbo decoding process by another decoder. In this algorithm, the soft-decision output is obtained by calculating the likelihood ratio

$$\Lambda(\hat{d}_k) = \frac{\sum_m \lambda_k^{1,m}}{\sum_m \lambda_k^{0,m}} \tag{4.14}$$

and

$$\lambda_k^{i,m} = P(d_k = i, S_k = m \mid R_1^N) \tag{4.15}$$

where $\lambda_k^{i,m}$ is the joint probability that at time instant k, data $d_k = i$ and state $S_k = m$, conditioned on the received binary sequence R_1^N, is observed from time $k = 1$ through some time N. R_1^N represents a received codeword sequence after it has been transmitted in the channel, demodulated, and presented to the decoder in soft output form.

In effect, the MAP algorithm requires that the output sequence from the demodulator be presented to the decoder as a block of N bits at a time. We can rewrite R_1^N in the form

$$R_1^N = \{R_1^{k=1}, R_k, R_{k+1}^N\} \tag{4.16}$$

To facilitate the use of Bayes' theorem, Equation (4.15) is partitioned using the letters *A, B, C, D* and is rewritten as

$$\lambda_k^{i,m} = P(\underbrace{d_k = i}_{A}, \underbrace{S_k = m}_{B} \mid \underbrace{R_1^{k-1}}_{B}, \underbrace{R_k}_{C}, \underbrace{R_{k+1}^N}_{D}) \tag{4.17}$$

Recall from Bayes' theorem that

$$P(A \mid B,C,D) = \frac{P(A,B,C,D)}{P(B,C,D)} = \frac{P(B \mid A,C,D)\, P(A,C,D)}{P(B,C,D)}$$

$$= \frac{P(B \mid A,C,D)\, P(D \mid A,C) P(A,C)}{P(B,C,D)} \tag{4.18}$$

The application of this theorem to Equation (4.17) yields

$$\lambda_k^{i,m} = P(R_1^{k-1} \mid d_k = i, S_k = m, R_k^N) P(R_{k+1}^N \mid d_k = i, S_k = m, R_k)$$

$$\times P(d_k = i, S_k = m, R_k) / P(R_1^N) \qquad (4.19)$$

where $R_k^N = \{R_k, R_{k+1}^N\}$ is the consolidated term of $P(C, D)$. The first numerator on the right side of Equation (4.19) is defined as the forward state metric at time k and state m, denoted as α_k^m.

$$\alpha_k^m = P(R_1^{k-1} \mid d_k = i, S_k = m, R_k^N) \qquad (4.20)$$

For R_1^{k-1}, $d_k = i$ and R_k^N are irrelevant since the assumption that $S_k = m$ implies that events before time k are not influenced by observations after time k. In other words, the past is not affected by the future. Thus $P(R_1^{k-1})$ is independent of the fact that $d_k = i$ and sequence R_k^N. However, since the encoder has memory, the encoder state $S_k = m$ is based on the past, so this term is relevant and must be left in the expression. Using these facts in Equation (4.20) we get

$$\alpha_k^m = P(R_1^{k-1} \mid S_k = m) \qquad (4.21)$$

Similarly, the second numerator on the right side of Equation (4.19) represents a reverse state metric β_k^m at time k and state m, described by

$$\beta_{k+1}^{f(i,m)} = P(R_{k+1}^N \mid d_k = i, S_k = m, R_k)$$

$$= P(R_{k+1}^N \mid S_{k+1} = f(i,m)) \qquad (4.22)$$

where $f(i,m)$ is the next state, given an input i and state m, and $\beta_{k+1}^{f(i,m)}$ is the reverse state metric at time $k+1$ and state $f(i,m)$. In other words, $\beta_{k+1}^{f(i,m)}$ is a probability at future time $k+1$, which depends on the state at time $k+1$ which in turn is $f(i,m)$, a function of the input bit and the state at current time k.

The third numerator on the right side of Equation (4.19) is defined as the branch metric at time k and state m, denoted as $\delta_k^{i,m}$ in the form

$$\delta_k^{i,m} = P(d_k = i, S_k = m, R_k) \tag{4.23}$$

With the definition of the three metrics in Equation (4.21) to (4.23), the joint probability $\lambda_k^{i,m}$ in Equation (4.19) can be expressed as

$$\lambda_k^{i,m} = \frac{\alpha_k^m \ \beta_{k+1}^{f(i,m)} \ \delta_k^{i,m}}{P(R_1^N)} \tag{4.24}$$

and the likelihood ratio in Equation (4.14) can be rewritten as

$$\begin{aligned} \Lambda(\hat{d}_k) &= \frac{\sum_m \lambda_k^{1,m}}{\sum_m \lambda_k^{0,m}} \\ &= \frac{\sum_m \alpha_k^m \ \beta_{k+1}^{f(1,m)} \ \delta_k^{1,m}}{\sum_m \alpha_k^m \ \beta_{k+1}^{f(0,m)} \ \delta_k^{0,m}} \end{aligned} \tag{4.25}$$

where $\Lambda(\hat{d}_k)$ is the likelihood ratio of the k-th data bit. In the following subsections, numerical calculation of these three metrics as well as a graphical representation will be given.

4.1.2.1 The Forward State Metric

Given the information of the previous decoded data bit $d_{k-1} = j$ and state $S_{k-1} = m'$, the forward state metric α_k^m can be represented as the summation of all possible transition probabilities from time k -1 to time k

$$\alpha_k^m = \sum_{m'} \sum_{j=0}^{1} P(\, d_{k-1} = j,\ S_{k-1} = m',\ R_1^{k-1} \mid S_k = m) \tag{4.26}$$

Rewriting R_1^{k-1} as $\{R_1^{k-2}, R_{k-1}\}$ we can get

$$\alpha_k^m = \sum_{m'} \sum_{j=0}^{1} P(\underbrace{d_{k-1} = j}_{A},\ \underbrace{S_{k-1} = m'}_{B},\ \underbrace{R_1^{k-2}}_{C},\ \underbrace{R_{k-1}}_{D} \mid S_k = m) \tag{4.27}$$

From Bayes' theorem

$$P(A, B, C \mid D) = \frac{P(A,B,C,D)}{P(D)}$$

$$= \frac{P(B \mid A,C,D)\, P(A,C,D)}{P(D)}$$

$$= P(B \mid A, C, D) \cdot P(A,C \mid D)$$

Thus, we can express Equation (4.26) as

$$\alpha_k^m = \sum_{m'} \sum_{j=0}^{1} P(\, R_1^{k-2} \mid d_{k-1} = j,\ S_{k-1} = m',\ S_k = m, R_{k-1})$$

$$\times P(\, d_{k-1} = j,\ S_{k-1} = m',\ R_{k-1} \mid S_k = m) \tag{4.28}$$

$$= \sum_{j=0}^{1} \underbrace{P(R_1^{k-2} \mid S_{k-1} = b(j,m))}_{\alpha_{k-1}^{b(j,m)}}\ \underbrace{P(d_{k-1} = j,\ S_{k-1} = b(j,m), R_{k-1})}_{\delta_{k-1}^{j,b(j,m)}} \tag{4.29}$$

where $b(j,m)$ is the state going backwards in time from state m, through the previous branch corresponding to input j. Since knowledge about the state m' and the input j at time k -1 can be obtained from the trellis diagram of the RSC encoder, Equation (4.29) can replace Equation (4.28) and be further simplified to

$$\alpha_k^m = \sum_{j=0}^{1} \alpha_{k-1}^{b(j,m)} \delta_{k-1}^{j,b(j,m)} \tag{4.30}$$

or

$$\alpha_k^m = \alpha_{k-1}^{b(0,m)} \cdot \delta_{k-1}^{0,b(0,m)} + \alpha_{k-1}^{b(1,m)} \cdot \delta_{k-1}^{1,b(1,m)} \tag{4.31}$$

This indicates that the forward state metric at time k and state m can be obtained by summing two weighted state metrics from time k -1. The weighting factor consists of the branch metrics associated with the transitions corresponding to data bits j = 0 and j = 1.

4.1.2.2 The Reverse State Metric

Starting from the metric $\beta_{k+1}^{f(i,m)} = P(R_{k+1}^N \mid S_{k+1} = f(i,m))$ in Equation (4.23), we can define the reverse state metric at time k and state m as

$$\beta_k^m = P(R_k^N \mid S_k = m) = P(R_k, R_{k+1}^N \mid S_k = m) \tag{4.32}$$

The metric β_k^m can be rewritten in terms of the summation of all possible transition probabilities to time k +1

$$\beta_k^m = \sum_{m'} \sum_{j=0}^{1} P(\underbrace{d_k = j, S_{k+1} = m'}_{A}, \underbrace{R_k}_{B}, \underbrace{R_{k+1}^N}_{C} \mid \underbrace{S_k = m}_{D}) \tag{4.33}$$

Apply Bayes' theorem again

$$P(A, B, C \mid D) = \frac{P(A,B,C,D)}{P(D)}$$

$$= \frac{P(C \mid A,B,D)\, P(A,B,D)}{P(D)}$$

$$= P(C \mid A, B, D) \cdot P(A,B \mid D)$$

we get

$$\beta_k^m = \sum_{m'} \sum_{j=0}^{1} P(R_{k+1}^N \mid S_k = m, d_k = j, S_{k+1} = m', R_k)$$

$$\times P(d_k = j, S_{k+1} = m', R_k \mid S_k = m) \tag{4.34}$$

Note that the next state $S_{k+1} = m'$ at time $k+1$ can be defined by the data bit $d_k = j$ and state $S_k = m$ at time k as a function $S_{k+1} = f(j,m)$. With this knowledge, the second term of Equation (4.34) is now rewritten in the form

$$\beta_k^m = \sum_{j=0}^{1} P(R_{k+1}^N \mid S_{k+1} = f(j,m))\, P(d_k = j, S_k = m, R_k) \tag{4.35}$$

which is equivalent to

$$\beta_k^m = \sum_{j=0}^{1} \beta_{k+1}^{f(j,m)}\, \delta_k^{j,m} \tag{4.36}$$

Equation (4.36) indicates that the reverse state metric at time k and state m is obtained by summing two weighted state metrics from time $k+1$. The weighting factor consists of the branch metrics associated with transitions corresponding to data bits $j = 0$ and $j = 1$.

4.1.2.3 The Branch Metric

From Equation (4.23) we have

$$\delta_k^{i,m} = P(d_k = i, S_k = m, R_k)$$

$$= P(R_k \mid d_k = i, S_k = m)\, P(S_k = m \mid d_k = i)\, P(d_k = i) \qquad (4.37)$$

where R_k represents the sequence $\{r_0, r_1\}$, r_0 is the corrupted data bit, and r_1 is the corresponding noisy received parity bit. Since the noise affecting the data and the parity are independent, the current state is independent of the current input and can therefore be any of the 2^v states, where v is the memory order of the RSC encoder. Thus we have

$$P(S_k = m \mid d_k = i) = \frac{1}{2^v}$$

and

$$\delta_k^{i,m} = P(r_0 \mid d_k = i, S_k = m)\, P(r_1 \mid d_k = i, S_k = m)\, \frac{\pi_k^i}{2^v} \qquad (4.38)$$

where π_k^i is equivalent to $P(d_k = i)$, the a priori probability of d_k. Since the received signal R_k is transmitted in an AWGN channel, the received sequence $\{r_0, r_1\}$ is of zero mean and variance σ^2. Thus we can replace the probability terms in Equation (4.38) with their probability density functions (pdf) as follows

$$P(r_0 \mid d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-\frac{1}{2}\left(\frac{r_0 - u_k^i}{\sigma}\right)^2\right] dr_0$$

$$P(r_1 \mid d_k = i, S_k = m) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left[-\frac{1}{2}\left(\frac{r_1 - v_k^{i,m}}{\sigma}\right)^2\right] dr_1$$

where u_k and v_k represent the transmitted data bits and parity bits, respectively, and dr_0 and dr_1 are the differentials of r_0 and r_1. Equation (4.38) can be rewritten in

the form

$$\delta_k^{i,m} = \frac{\pi_k^i}{2^\nu}\left\{\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left[-\frac{1}{2}\left(\frac{r_0 - u_k^i}{\sigma}\right)^2\right]dr_0\right\}\left\{\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left[-\frac{1}{2}\left(\frac{r_1 - v_k^{i,m}}{\sigma}\right)^2\right]dr_1\right\}$$

$$= A_k\,\pi_k^i\,\exp\left[\frac{1}{\sigma^2}\left(r_0\,u_k^i + r_1\,v_k^{i,m}\right)\right] \qquad (4.39)$$

Note that A_k is a constant that absorbs the terms dr_0, dr_1, $\frac{1}{2^\nu}$, and $\frac{1}{2\pi}$. Substitute Equation (4.39) into Equation (4.28), the likelihood ratio is

$$\Lambda(\hat{d}) = \pi_k\,\exp\left(\frac{2\,r_0}{\sigma^2}\right)\frac{\sum_m \alpha_k^m \exp\left(\frac{r_1\,v_k^{1,m}}{\sigma^2}\right)\beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \exp\left(\frac{r_1\,v_k^{0,m}}{\sigma^2}\right)\beta_{k+1}^{f(0,m)}}$$

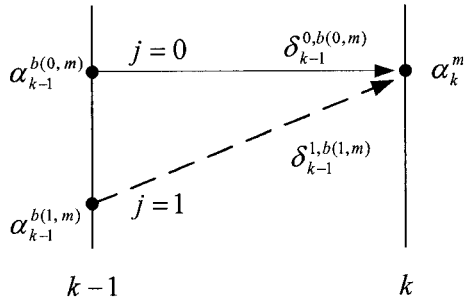
$$= \pi_k\,\exp\left(\frac{2\,r_0}{\sigma^2}\right)\pi_k^e \qquad (4.40)$$

where $\pi_k = \pi_k^1 / \pi_k^0$ is the input a priori probability ratio, and π_k^e is the output extrinsic likelihood ratio, all at time k, and can be written in the form

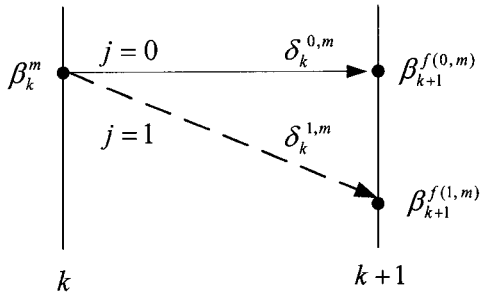
$$\pi_k^e = \frac{\sum_m \alpha_k^m \exp\left(\frac{r_1\,v_k^{1,m}}{\sigma^2}\right)\beta_{k+1}^{f(1,m)}}{\sum_m \alpha_k^m \exp\left(\frac{r_1\,v_k^{0,m}}{\sigma^2}\right)\beta_{k+1}^{f(0,m)}} \qquad (4.41)$$

Figure 4.2 is the graphical representation of how the state metrics are calculated [31].

(a) Forward State Metric



(b) Reverse State Metric



$$\alpha_k^m = \alpha_{k-1}^{b(0,m)} \cdot \delta_{k-1}^{0,b(0,m)} + \alpha_{k-1}^{b(1,m)} \cdot \delta_{k-1}^{1,b(1,m)}$$

$$\beta_k^m = \beta_{k+1}^{f(0,m)} \cdot \delta_k^{0,m} + \beta_{k+1}^{f(1,m)} \cdot \delta_k^{1,m}$$

(c) Branch Metric $\quad \delta_k^{i,m} = A_k \, \pi_k^i \exp\left[\frac{1}{\sigma^2}\left(r_0 \, u_k^i + r_1 \, v_k^{i,m}\right)\right]$

Figure 4.2: Graphical Representation of State Metrics Calculation

4.2 Log-MAP Algorithm

By taking the logarithm of Equation (4.40) [27] we get

$$L(\hat{d}_k) = L(\pi_k) + \frac{2\,r_0}{\sigma^2} + L(\pi_k^e)$$

$$= L(d_k) + L_C(r) + L_e(\hat{d}_k) \qquad (4.42)$$

where $L(d_k)$ is the a priori probability, equivalent to $L(\pi_k)$, and $L_c(r)$ is the LLR of channel measurement, and it can be calculated by taking the logarithm of the second term of Equation (4.40).

$$L_C(r) = \log_e\left[\exp\left(\frac{2\,r_0}{\sigma^2}\right)\right] = \frac{2\,r_0}{\sigma^2} \qquad (4.43)$$

The term $L_e(\hat{d}_k)$ represents the extrinsic LLR of the estimate bit $\hat{d}_k$, and is

equivalent to $L(\pi_k^e)$, the LLR of the output extrinsic likelihood. Equation (4.42) is exactly the same as the result in Equation (4.13) in Section 4.1.1.

The reason for taking the logarithm is to reduce the complexity. The MAP algorithm can be implemented in terms of a likelihood ratio, however, implementation using likelihood ratios is very complex because of the multiply operations that are required. By implementing the MAP algorithm in the logarithmic domain, the complexity can be greatly reduced since the multiply operations are replaced with addition.

4.3 Soft Output Viterbi Algorithm

In 1967, Viterbi proposed a decoding algorithm for convolutional codes that has become known as the Viterbi algorithm (VA) [32]. The algorithm was recognized by Forney to be a maximum likelihood decoder [33]. Essentially, the algorithm performs estimation of the input sequence of a discrete-time finite-state Markov process observed in memoryless noise.

The VA is an efficient way of finding a path in the trellis with a minimum distance from the maximum likelihood (ML) path. It is based on the idea that among the paths merging into a state in the code trellis, only the most probable path need to be saved for future processing, while all other paths can be discarded with no consequence to achieve optimal decoding. Due to the property that only one path will be kept to enter a node, there will be only one path left at the end of the decoding process. This path is called the survivor path.

The decision regarding the transmitted information sequence d is made at the final time instant. The decoder will select the binary sequence $\hat{d}$ corresponding to

this survivor path as the hard estimate of the information sequence d.

The problem with the Viterbi Algorithm is that it generates hard decisions resulting in performance deficiency in multistage (iterative) decoding. To overcome this problem, an algorithm called soft output Viterbi algorithm (SOVA) was proposed to produce soft outputs [34].

The SOVA estimates the soft output for each transmitted binary symbol in the form of the log-likelihood function

$$L(d \mid r) = \log\left[\frac{P(d=1 \mid r)}{P(d=0 \mid r)}\right] = L(\hat{d}) \tag{4.7}$$

where r is the received sequence and $P(d=i \mid r)$ is the a posteriori probability of the transmitted symbol i. The SOVA decoder makes a hard decision by comparing the $L(\hat{d})$ to a threshold equal to zero so that

$$\hat{d} = \begin{cases} 1 & \text{if } L(\hat{d}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.44}$$

To generate the soft output, first we define a branch metric $\gamma_k^{(p)}$ and a path metric $\mu_k^{(p)}$ as follows

$$\gamma_k^{(p)} = \sum_{i=0}^{n-1} \left(r_{k,i} - x_{k,i}\right)^2 \tag{4.45}$$

$$\mu_k^{(p)} = \sum_{k'=1}^{k} \gamma_{k'}^{(p)} = \mu_{k-1}^{(p)} + \gamma_k^{(p)} \tag{4.46}$$

Here, p denotes the survivor path in the code trellis, $r_{k,i}$ is the received data bit, $x_{k,i}$ is the modulated information data bit, and the branch metric $\gamma_k^{(p)}$ is the Euclidean

distance assigned to each branch on the path p at time instant k. The path metric $\mu_k^{(p)}$ at time k can be expressed as the summation of the path metric $\mu_{k-1}^{(p)}$ at time $k-1$ with the branch metric $\gamma_k^{(p)}$ at time k. Note that the branch metric in the SOVA is not the same as the one defined in the BCJR algorithm in Section 4.1.2.

The SOVA decoder selects the path p with the minimum path metric $\mu_{k,\min}$ as the maximum likelihood (ML) path in the same way as the standard Viterbi algorithm. The probability of selecting this path is approximately expressed in the form

$$P(d \mid r) = P(survivor\ path\ p \mid r) \sim \exp(-\mu_{k,\min}) \tag{4.47}$$

If we denote by $\mu_{k,C}$ the complementary symbol to the ML symbol at time k and assume the ML symbol at time k is 1 (its complementary symbol is 0), then we have

$$P(d=1 \mid r) \sim \exp(-\mu_{k,\min}) \tag{4.48}$$

and

$$P(d=0 \mid r) \sim \exp(-\mu_{k,C}) \tag{4.49}$$

The logarithm of the ratio of the above two probabilities is given by

$$\begin{aligned} \log\frac{P(d=1 \mid r)}{P(d=0 \mid r)} &\sim \log\frac{\exp(-\mu_{k,\min})}{\exp(-\mu_{k,C})} \\ &= \mu_{k,C} - \mu_{k,\min} \\ &= \mu_k^0 - \mu_k^1 \end{aligned} \tag{4.50}$$

where μ_k^1 represents the minimum path metric for all paths for which d is 1, and μ_k^0 is

the case when d is 0. On the other hand, if the maximum likelihood symbol at time k is 0 (its complementary symbol is 1), giving $\mu_{k,\min} = \mu_k^0$ and $\mu_{k,C} = \mu_k^1$, then we can get the log-likelihood ratio as

$$\log \frac{P(d=1|r)}{P(d=0|r)} \sim \log \frac{\exp(-\mu_{k,C})}{\exp(-\mu_{k,\min})}$$

$$= \mu_{k,\min} - \mu_{k,C} = \mu_k^0 - \mu_k^1 \tag{4.51}$$

As shown in Equation (4.50) and (4.51), regardless of the value of the ML hard estimate, the log-likelihood ration can be expressed in the form

$$L(\hat{d}) = \log \frac{P(d=1|r)}{P(d=0|r)}$$

$$\sim \mu_k^0 - \mu_k^1 \tag{4.52}$$

In other words, the soft output of the decoder can be obtained as the difference of the minimum path metric among all the paths with symbol 0 at time k and the minimum path metric among all the paths with symbol 1 at time k. As in the BCJR algorithm, the sign of $L(\hat{d})$ denotes the hard decision and its magnitude provides the reliability that can be used for decoding in the next stage.

4.4 Iterative Decoding of Turbo Codes

Based on previous discussion, we know that the soft decoder output $L(\hat{d})$ is a real number that provides a hard decision as well as the reliability of that decision. That is, the sign of $L(\hat{d})$ denotes the hard decision, and the magnitude of $L(\hat{d})$ denotes the reliability of that decision. The value of $L_e(d)$ often has the same sign as $L_c(r) + L(d)$ and $L_e(d)$ is independent of the input of the other decoder, thus the

extrinsic informaion $L_e(d)$ acts to improve the reliability of $L(\hat{d})$. Figure 4.3 shows the iterative decoding process of a soft-in-soft-out decoder.

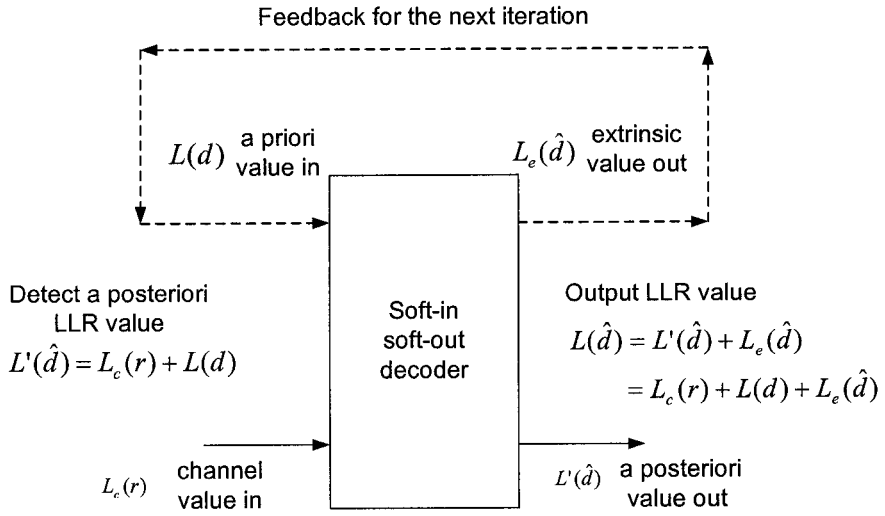


Figure 4.3: Soft-in-soft-out Decoding Scheme

To distinguish the LLRs of Decoder 1 and 2, different notations are given as follows. Recall Equation (4.42)

$$L(\hat{d}_k) = L(d_k) + L_c(r) + L_e(\hat{d}_k) \tag{4.53}$$

which we rewrite it in the form

$$L_p^{D1} = L_{a1} + L_{c1} + L_e^{D1} \tag{4.54}$$

where L_p^{D1} is the a posteriori LLR of Decoder 1, and is $L(\hat{d}_k)$ in the previous discussion. Similarly, L_{a1} is the a priori LLR, and equivalent to the extrinsic information comes from Decoder 2 during the iterative decoding process. Also, L_{c1} is the a posteriori LLR of the channel measurement, denoted as $L_c(r)$ in Equation (4.53). L_e^{D1} is the extrinsic information LLR, and it is the term $L_e(\hat{d}_k)$ in Equation

(4.53). Note that $L_p^{D2'}$, L_{c2}', L_{a2}', and $L_e^{D2'}$ represent the interleaved sequences.

The iterative decoding process consists of the following steps:

(1) Initialize extrinsic LLRs $L_e^{D1(0)}$=0 and $L_e^{D2(0)'}$=0.

(2) For iterations $I = 1, 2, \Lambda, N_{\max}$ where $N_{\max}$ is the total number of iterations

- Compute $L_p^{D1(I)}$ and $L_p^{D2(I)'}$ by using Log-MAP algorithm in Equation (4.42) or SOVA algorithm in Equation (4.52)

- Compute $L_e^{D1(I)}$ as

$$L_e^{D1(I)} = L_p^{D1(I)} - \frac{2\,r_0}{\sigma^2} - L_e^{D2(I-1)} \tag{4.55}$$

- Compute $L_e^{D2(I)'}$ as

$$L_e^{D2(I)'} = L_p^{D2(I)'} - \frac{2\,r_0'}{\sigma^2} - L_e^{D1(I-1)'} \tag{4.56}$$

(3) After $N_{\max}$ iterations, make a hard decision of $\hat{d}$ based on the deinterleaved soft output $L_e^{D2(N_{\max})}$

Figure 4.4 illustrates the iterative decoding structure of the turbo decoder.

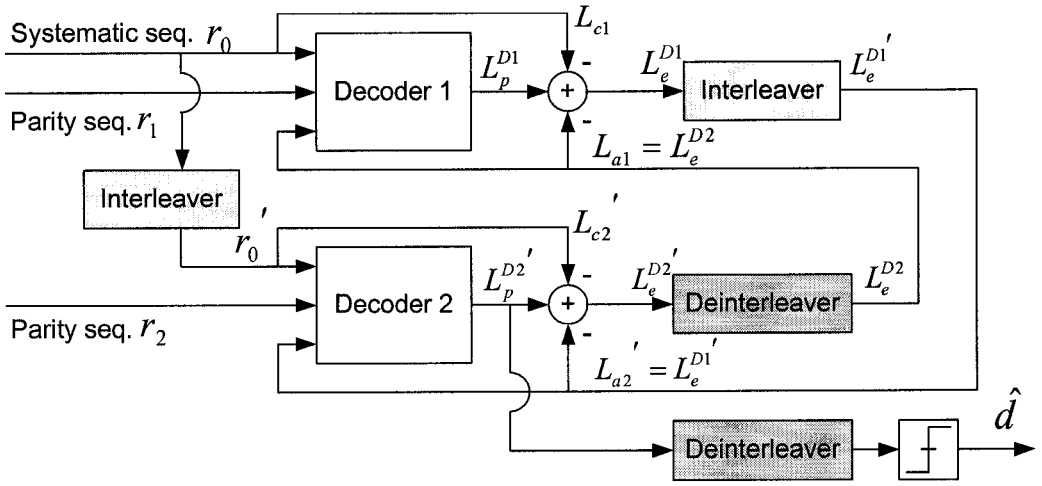


Figure 4.4: Iterative Decoding Schematic of Turbo Decoder

In a binary symmetric channel (BSC), the probability of error for binary symbols 0 and 1 are the same. The LLR of the a priori probability $L(d)$ is set to zero initially. During the iterative decoding process, the extrinsic LLR will be taken into account as $L(d)$ for the other decoder. Notice that the extrinsic information serves as a refinement of the a priori probability of the data for the next iteration.

4.5 Stopping Rules for Turbo Decoders

Decoders of turbo codes are iterative in nature, and a certain number of iterations are required before reaching a satisfactory degree of confidence regarding a frame to be decoded. Most standard turbo decoders have used a fixed number of iterations. Recent researches [35, 36] provide some alternative criteria for more efficient stopping rules that significantly increase the average decoding speed without sacrificing decoder performance. In this section, four simple criteria will be introduced briefly. Of the four, the cyclic redundancy check (CRC) code is adopted in later simulations to speed up the iterative decoding process.

In general, Hard-decision rules attempt to detect unreliable decoded sequences by evaluating the tentative decoded bits (hard decisions) at the end of each iteration. Soft-decision rules are based on comparing a metric on bit reliabilities (soft decisions) with a threshold. A CRC rule detects unreliable decoded sequences using an outer cyclic redundancy check (CRC) code applied to hard decoded bits.

4.5.1 Hard Bit Decisions

The hard-decision rule simply checks whether identical tentative bit decisions are made at successive iterations or half-iterations. With this rule, after each iteration both component decoders make tentative decoded bit decisions, and the iterative process is stopped at the earliest iteration, $n \leq N_{\max}$, when the two decoders completely agree, that is,

$$u_{i,1}^{n} = u_{i,2}^{n}, \quad \forall i, 1 \leq i \leq K \tag{4.57}$$

where $u_{i,1}^{n}$ and $u_{i,2}^{n}$ are the i-th decoded bits from the first and second decoders, respectively, at iteration n, and K is the information block size. Therefore, agreement on decoded sequences at each iteration will cause iterative process to terminate and the current decoded sequence to be sent to the output of the turbo decoder. This rule has been introduced in [37].

4.5.2 Soft Bit Decisions

Soft decision rules are based on comparing a metric on bit reliability with a threshold. Two simple metrics that can be used to decide when to stop the iterative decoding process are the average and minimum bit reliabilities. At each iteration, the turbo decoder computes a metric on the log-likelihoods of the information bits and compares it with a preset threshold value. If the metric is smaller than the threshold,

the decoder continues with a new iteration; otherwise if the metric is larger than the threshold, the decoder stops.

Let $\lambda_{i,1}^n$ denotes the i-th bit reliability from the first decoder and $\lambda_{i,2}^n$ denote the i-th bit reliability from the second decoder at iteration n. The average of the absolute values of the bit reliabilities from one of the decoders is compared with a threshold θ_1. The rule is satisfied at the earliest iteration $n \le N_{\max}$ such that

$$\frac{1}{K}\sum_{i=1}^{K} | \lambda_{i,2}^n | \ge \theta_1 \tag{4.58}$$

In this example, the reliability from the second decoder is compared to the threshold θ_1. Another soft decision is made by comparing the minimum on the absolute value of the bit reliability. The decoder will stop if the following condition is satisfied

$$\min_{0<i\le K} | \lambda_{i,2}^n | \ge \theta_2 \tag{4.59}$$

4.5.3 Cyclic Redundancy Check (CRC) Rule

This error detection rule is a stopping rule based on detecting erroneous decoded sequences using an outer cyclic redundancy check (CRC) code applied to hard-decoded bits. A separate error detection code, such as CRC code, can be concatenated as an outer code with an inner turbo code. The condition for stopping with this rule is satisfied whenever the syndrome of the CRC code is zero. This syndrome comes from the remainder of the long division which divides the decoded sequence by the CRC code generating polynomial. Notice that when CRC stopping rule is applied, both undetected and falsely detected errors may occur. A more reliable stopping rule is to detect two or more consecutive zero-syndromes.

The CRC encoder can be implemented by using shift registers. Figure 4.4 shows the structure of the CRC encoder, and the syndrome is the data in the shift

registers.

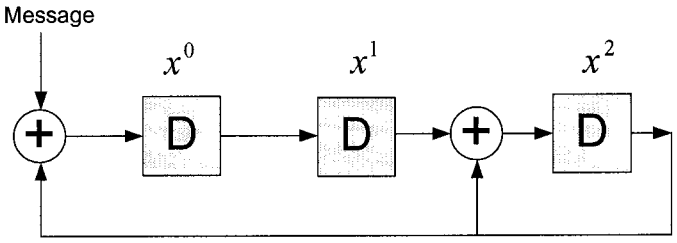


Figure 4.5: CRC Calculation using Shift Register

In Figure 4.5, the code generating polynomial is $x^3 + x^2 + 1$ and modulo 2 addition is performed at the input of the x^0 and x^2 position in the encoder structure.

4.5.4 Finite Termination Condition

The finite termination condition for all of the stopping rules discussed above is an adjunct stopping condition that ceases the decoding after a maximum number of $N_{\max}$ iterations. This condition prevents an endless loop if the stopping rule is never satisfied. The stopping rules discussed here can increase the average decoding speed and reduce an excessive overhead without sacrificing the performance.

In summary, this chapter introduced two turbo decoding algorithms and presented some theoretical derivations. The Log-MAP algorithm is a modified version of the BCJR algorithm, and the former reduces a certain degree of complexity while its performance is almost identical with the optimum MAP decoding. The low-complexity SOVA is a suboptimum approximation of the MAP algorithm, however, where a coding loss of ~0.5 dB is introduced with regard to Log-MAP algorithm. An iterative decoding process, schematically illustrated in Figure 4.4, was introduced and discussed, together with some stopping criteria that serve to increase average decoding speed.

Chapter 5 The Tuning Factor Effect

The discovery of the so called tuning factor lies in the problem solving of an error performance fluctuation phenomenon that occurs in turbo decoding process. This fluctuation appears more frequently in the Multiple Turbo Coding [38, 39] where the power of the transmitted signal is very low. Instead of having two component RSC encoders in the structure, multiple turbo codes use three or more constituent encoders to encipher the information sequence to be transmitted, hence the name.

The feature characteristic of multiple turbo codes is that with a lower code rate, the error performance is better than the regular turbo codes with two encoders. The better performance can be achieved at the expense of larger number of iterations and higher decoding complexity. In other words, multiple turbo codes are hard to apply in a real-time communication system. A main beneficiary of multiple turbo codes is deep space communications where the transmitters are limited to be low power. In order to extract better quality data from the low power signals, large antennas are used to receive the transmitted signal. And high complexity (more computations) decoding techniques, such as multiple turbo codes, can be applied to decipher the highly corrupted signals of deep space communications without requiring the decoding process to be in real-time.

In the following sections, the fluctuation phenomenon will be described in detail. Techniques used to mitigate this phenomenon will be introduced, including the tuning factor method which not only solves the stability problem of turbo codes, but also improves the overall error performance.

5.1 The Bit Error Rate Fluctuation Phenomenon

The bit error rate fluctuation phenomenon is an abnormal incidence which occurs mostly when the signal to noise ratio (SNR) is low. During the iterative decoding process at receiver side, the bit error rate (BER) can sometimes increase instead of get improved. This phenomenon was mentioned in the last section of Berrou and Glavieux's original turbo codes paper [9].

Table 5.1 is an example of the bit error rate fluctuation phenomenon observed during multiple turbo coding simulation. Only one frame was transmitted. The frame size is 4096, g=(1, 5/7), rate 1/4, pseudo-random interleaver, and log-MAP algorithm is used.

Table 5.1: Bit Error Rate Fluctuation Phenomenon Example

# of iteration	1	2	3	4	5	6
Bit error rate	1.35e-01	7.08e-02	2.05e-02	4.88e-04	0.00e+00	0.00e+00
# of iteration	7	8	9	10	11	12
Bit error rate	0.00e+00	1.46e-02	4.06e-01	2.15e-01	1.24e-01	6.69e-02
# of iteration	13	14	15	16	17	18
Bit error rate	1.73e-02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	4.20e-02
# of iteration	19	20	21	22	23	24
Bit error rate	4.15e-01	2.19e-01	1.35e-01	6.49e-02	1.58e-02	0.00e+00

For the first 4 iterations, the bit error rate decreases and all errors are eliminated at the 5th iteration. However, as the decoding process continues, bit errors appear suddenly at iteration 8. Deterioration can be observed from iteration 8 to 11. And

from iteration 12 to 14, the result gets improved again. Notice that Table 5.1 shows only the results from iteration 1 to 24. The bit error rate fluctuation occurs through all the 100 iterations for this error sequence.

5.2 Stabilization Factor

To overcome the fluctuation phenomenon, Berrou and Glavieux provided a method by constraining the extrinsic information [9, 11]. They suggested at low SNR, the extrinsic information z_k be divided by $[1+\theta\cdot|L(\hat{d})|]$, where θ acts as a stabilization factor and $|L(\hat{d})|$ is the absolute value of the soft output of the decoder. A value $\theta = 0.15$ was adopted after several simulation tests at E_b/N_0 = 0.7 dB. Often the extrinsic information has the same sign as the sum of channel measurement and a priori probability, and hence acts to improve the reliability of $L(\hat{d})$.

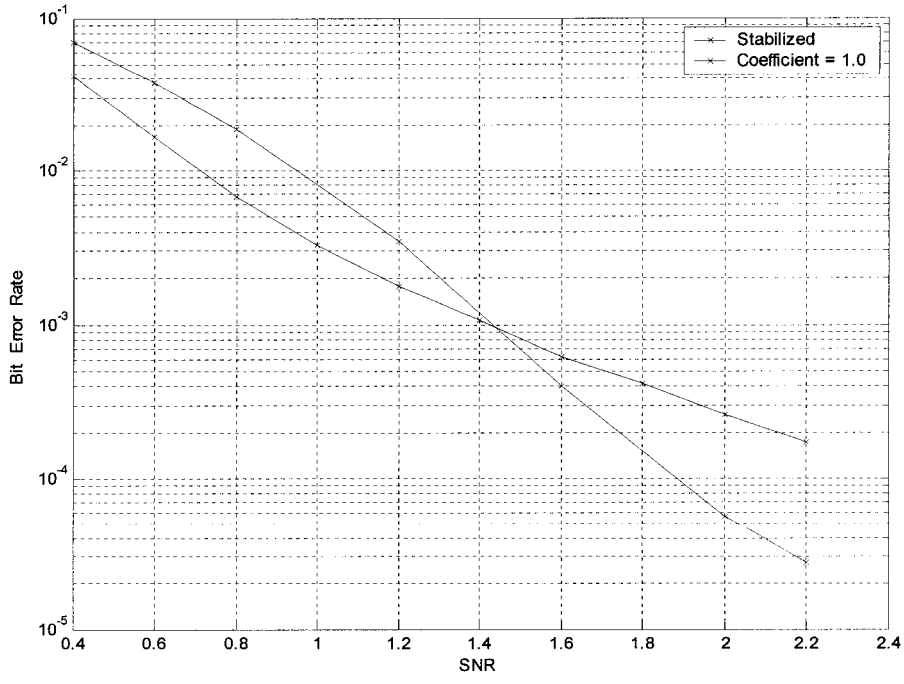


Figure 5.1: Comparison of Error Performance with and without Stability Factor

Figure 5.1 illustrates a comparison of error performance. The red line is the result without using stabilization factor, and the blue line represents the stabilized result. From the diagram, it is obvious that the stability factor not only removes the bit error fluctuation phenomenon, but also improves the error performance at low SNR's. Or equivalently, using the stability factor makes it possible to achieve the same error performance with less iterations.

Why does the stability factor reduce the bit error fluctuation? To answer this question, we have to look at the intermediate value of the extrinsic information and the soft output of the decoder among iterations. The decision-making rule of the MAP algorithm has been introduced in Chapter 4, where the sign of the soft output leads to the hard decision: if $L(\hat{d})>0$ decide d = +1, otherwise decide d = -1. The magnitude of $L(\hat{d})$ denotes the reliability of that decision.

The fluctuation phenomenon happens because the magnitude of the soft output at the error bit is small and very close to 0. During the decoding process, the extrinsic information will bring the soft output cross the "0" boundary back and forth, and hence the fluctuation. Thus to reduce this fluctuation phenomenon, one needs to reduce the magnitude of the extrinsic information.

Why can the stability factor accelerate the convergence speed? Let's look deeper into the decoding process. As mentioned before, the extrinsic information is the element that corrects the soft output value during iterative decoding process. Due to the effect of extrinsic information, the soft output will change its value iteration by iteration. If the signal is badly contaminated, then it takes more iterations for the extrinsic information to modify the soft output. On the other hand, if the received

signal isn't that erroneous, it takes less iteration to make the soft output come back to the right state.

Dividing the extrinsic information by $[1+\theta\cdot|L(\hat{d})|]$ reduces the soft output value in the decoding process. Although this will lead to a decrease in improvement, the overall effect is positive. That is, reducing the value of the extrinsic information can reduce the chance that the soft output crosses the "0" boundary. Therefore it can reduce the occurrence of bit errors and fluctuation.

The disadvantage of using this stability factor is that the improvement in error performance decreases while the number of iteration goes up. This is because the shift of soft output is shrinking if we keep dividing the extrinsic information. Figure 5.2(a) shows the bit error rate when a stability factor is used, and 5.2(b) shows results when a regular turbo code is used.

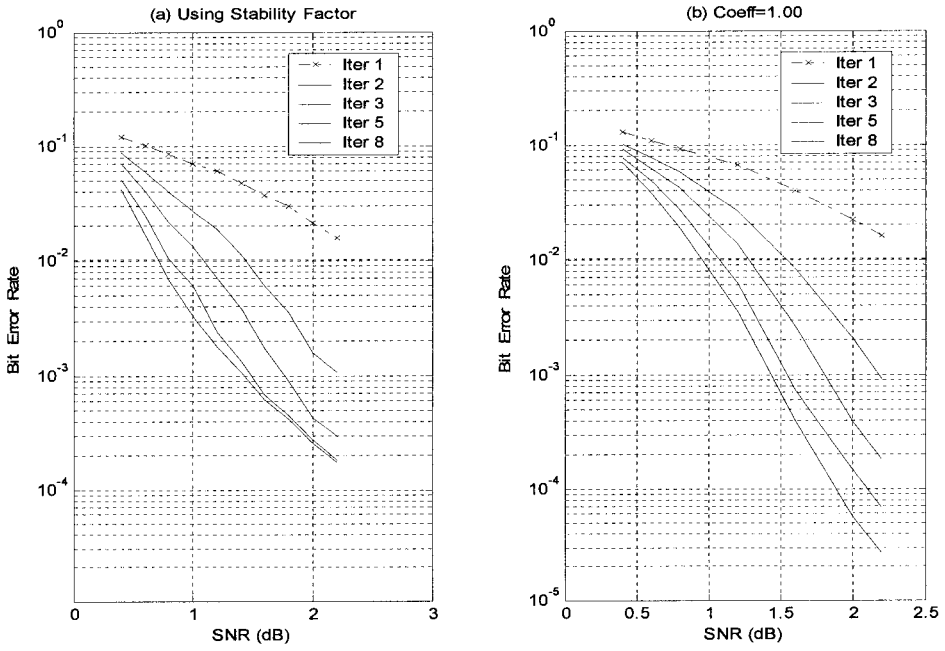


Figure 5.2: Comparison of Bit Error Performance (a) Using Stability Factor (b)Coefficient = 1.00

Notice that at high SNR, the error performance will be better if the decoder doesn't use the stability factor method. Notice also that n Figure 5.2(a), there is very little improvement from iteration 5 to iteration 8 when stability factor is applied. Figure 5.2(b) is the regular result without modifying the extrinsic information.

5.3 The Discovery of the Tuning Factor

Based on the previous discussion and example, a new approach is proposed here for reliable decoding of highly corrupted received signals. The approach relies on the premise that by scaling the magnitude of extrinsic information, the error performance and convergence speed of the decoding process will change.

Coincidently, when initial multiple turbo coding simulations were performed by the author, a stability factor wasn't used. This is because little published work explicitly addresses the fluctuation phenomenon. The phenomenon was only briefly mentioned in the last section of Berrou's original turbo codes paper [9, 11]. The persistence of the phenomenon in multiple turbo coding simulation results forced the author to search for other ways to address the problem. Fortunately, a clue was found in D. Divsalar and F. Pollara's multiple turbo coding report [38].

Instead of being divided by a stability factor, Divsalar and Pollara proposed multiplication of the extrinsic information by a coefficient "α" smaller than 1. In their report, the gain "α" was actually set to be equal to 1. They noticed experimentally that better convergence can be obtained by optimizing this gain factor for each iteration, starting from a value slightly less than 1 and then increasing the value toward one with increasing number of iterations.

From Equation (4.13), the log-likelihood ratio of the soft output decision $L(\hat{d})$ can be obtained from

$$L(\hat{d}) = L_c(r) + L(d) + L_e(\hat{d}) \tag{5.1}$$

After calculating the $L(\hat{d})$ using the BCJR or SOVA algorithm, the extrinsic LLR is calculated from

$$L_e(\hat{d}) = L(\hat{d}) - L_c(r) - L(d) \tag{5.2}$$

to which we apply a coefficient α to obtain

$$\alpha \cdot L_e(\hat{d}) = \alpha \cdot \left\{ L(\hat{d}) - L_c(r) - L(d) \right\} \tag{5.3}$$

The α coefficient takes into account the fact that the standard deviation of samples in matrix $L(\hat{d})$ and the matrix $L_e(\hat{d})$ are different. The standard deviation of the extrinsic information is very high in the first decoding steps and decreases as we iterate the decoding. The α coefficient is also used to reduce the effect of the extrinsic information in the soft decoder in the first decoding steps when the bit error rate is relatively high.

Note that the soft output changes bit wisely because the sequence $\hat{\mathbf{d}}$ is an estimate of the transmitted information. Thus, when the dividing stability factor $[1+\theta \cdot | L(\hat{d}) |]$ is applied, it will change its value bit by bit. Unlike stability factor, the coefficient α is merely a scalar. This implies the extrinsic information corresponding to the received bit stream will be scaled up or down in the same proportion at every bit position.

In the next section, decoding results using this α coefficient will be presented. And on the basis that this "α " coefficient is actually tunable and time-varying with the

channel, we then give "α" the name tuning factor T. Hence Equation (5.2) now assumes the form

$$T \cdot L_e(\hat{d}) = T \cdot \left\{ L(\hat{d}) - L_c(r) - L(d) \right\} \tag{5.4}$$

5.4 Example Single Frame Decoding using Tuning Factors

To investigate the effect of applying the "T" factor, one approach is to compare results obtained based on decoding a single frame. Since both the information sequence and the received sequence that produces the fluctuation phenomenon depicted in Table 5.1 are already available, multiple turbo coding simulations and comparisons can be performed. The interleaving map that permutes the systematic information is used in decoding the corresponding sequence.

The error sequence was generated under the condition that: three symmetric constituent encoders with G=[1, 5/7], unpunctured, rate = 1/4, 4096 bits/frame, pseudo-random interleaver, with Log-MAP decoding algorithm, under additive white Gaussian noise (AWGN) channel. Since there is only one frame being decoded, the comparison was made to investigate convergence speed. For the comparison of overall error performance, Monte Carlo simulation have to be performed. Results of Monte Carlo simulation will be given in the next Chapter.

Table 5.2 to 5.10 shows results of the first 16 iterations for each different tuning factor in the range from 0.25 to 1.10. The blue colored numbers represent the bit error rate improvement when the number of iteration increases. The red colored numbers represent the fluctuation phenomenon, where the bit error rate does not improve.

*Table 5.2: Tuning Factor **T=1.10** (Fluctuated)*

Iter #	1	2	3	4	5	6	7	8
BER	1.39e-01	7.35e-02	2.12e-02	4.88e-04	0.0e+00	0.0e+00	1.23e-01	4.50e-01
Iter #	9	10	11	12	13	14	15	16
BER	3.51e-01	1.98e-01	1.16e-01	6.27e-02	1.53e-02	7.32e-04	0.0e+00	4.88e-04

*Table 5.3: Tuning Factor **T=1.00** (Fluctuated)*

Iter #	1	2	3	4	5	6	7	8
BER	1.36-01	7.08e-02	2.05e-02	4.88e-04	0.0e+00	0.0e+00	0.0e+00	1.46e-02
Iter #	9	10	11	12	13	14	15	16
BER	4.06e-01	2.15e-01	1.24e-01	6.69e-02	1.73e-02	0.0e+00	0.0e+00	0.0e+00

*Table 5.4: Tuning Factor **T=0.90** (Fluctuated)*

Iter #	1	2	3	4	5	6	7	8
BER	1.36e-01	7.84e-02	2.54e-02	2.19e-03	0.0e+00	0.0e+00	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	1.24e-02	4.45e-01	2.66e-01	1.52e-01	7.76e-02	2.19e-02	2.44e-03	0.0e+00

Table 5.5: Tuning Factor T=0.85 (Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.37e-01	7.96e-02	2.73e-02	2.68e-03	0.0e+00	0.0e+00	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	1.95e-03	2.56e-01	2.76e-01	1.52e-01	8.72e-02	2.95e-02	3.66e-03

Table 5.6: Tuning Factor T=0.80 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.40e-01	8.35e-02	3.24e-02	4.39e-03	0.0e+00	0.0e+00	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.7: Tuning Factor T=0.75 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.42e-01	8.84e-02	4.10e-02	8.06e-03	0.0e+00	0.0e+00	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.8: Tuning Factor T=0.70 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.45e-01	9.52e-02	5.44e-02	1.95e-02	2.19e-03	0.0e+00	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.9: Tuning Factor T=0.65 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.49e-01	1.02e-01	6.96e-02	3.22e-02	8.54e-03	4.88e-04	0.0e+00	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.10: Tuning Factor T=0.60 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.52e-01	1.10e-01	8.10e-02	5.56e-02	2.95e-02	8.06e-03	4.88e-04	0.0e+00
Iter #	9	10	11	12	13	14	15	16
BER	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.11: Tuning Factor T=0.50 (Not Fluctuated)

Iter #	1	2	3	4	5	6	7	8
BER	1.59e-01	1.26e-01	1.12e-01	9.99e-02	9.33e-02	8.32e-02	7.62e-02	7.13e-02
Iter #	9	10	11	12	13	14	15	16
BER	6.44e-02	5.88e-02	4.90e-02	4.03e-02	3.00e-02	2.05e-02	1.09e-02	3.41e-03
Iter #	17	18	19	20	21	22	23	24
BER	4.88e-04	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00	0.0e+00

Table 5.12: Tuning Factor T=0.45 (Can't Decode)

Iter #	1	2	3	4	5	6	7	8
BER	1.59e-01	1.32e-01	1.24e-01	1.20e-01	1.17e-01	1.14e-01	1.13e-01	1.12e-01
Iter #	9	10	11	12	13	14	15	16
BER	1.12e-01	1.12e-01	1.12e-01	1.12e-01	1.12e-01	1.12e-01	1.12e-01	1.12e-01

Table 5.13: Tuning Factor T=0.40 (Can't Decode)

Iter #	1	2	3	4	5	6	7	8
BER	1.63e-01	1.39e-01	1.31e-01	1.30e-01	1.28e-01	1.28e-01	1.28e-01	1.28e-01
Iter #	9	10	11	12	13	14	15	16
BER	1.28e-01	1.28e-01	1.28e-01	1.28e-01	1.28e-01	1.28e-01	1.28e-01	1.28e-01

Table 5.14: Tuning Factor T=0.25 (Can't Decode)

Iter #	1	2	3	4	5	6	7	8
BER	1.72e-01	1.66e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01
Iter #	9	10	11	12	13	14	15	16
BER	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01	1.64e-01

From the result above, we can see that by changing the tuning factor T, the convergence speed in the decoding process is affected dramatically. In this example, this error sequence can be best decoded by using T=0.75 or 0.80. The bit error can be totally eliminated at the 5^{th} iteration. On the other hand, when the tuning factor is chosen to be below 0.50, the sequence can't be decoded at all. Notice that when T=1.10, the sequence can be decoded but the bit error fluctuation occurs.

The tuning factors T=0.70, 0.75, and 0.80 are factors that lead to good performance in the decoding process. However, when simulations are performed

repeatedly, we found that some sequences can't be decoded by any of these factors. Instead, those undecodable sequences can surprisingly be decoded by tuning factors with different value. In other words, different error patterns can be successfully decoded by applying the appropriate factor. Since the factor is tunable in nature, we give it the name - "Tuning Factor."

The results of applying a tuning factor to the single frame multiple turbo coding simulation reflect some new properties of turbo codes. First, we see that by applying an appropriate tuning factor, the bit error rate fluctuation can be alleviated. Second, the error performance is better than the result using the stability factor. Moreover, examples have been found in simulations that in the case of a time-varying channel, the value of the tuning factor is also time-varying.

In the following chapter, simulations designed to unveil other properties about tuning factor will be conducted. A different type of error pattern analysis is given, and a more complex decoder scheme using the tuning factor is implemented.

Chapter 6 Simulation Results and Analysis

In this chapter, results of Monte Carlo simulations based on two types of simulation schemes are presented. The first one examines the effect of the tuning factor and compares results when different tuning factors are applied. In the error pattern analysis, error sequences that can't be decoded are recorded and decoded again by applying tuning factors with different value. It is shown that successful decoding can be achieved and optimized (in shortest number of iteration) if appropriate factors can be found.

The second simulation implements a new decoding scheme based on the new properties of turbo codes when a tuning factor is used. In the new decoding structure, the decoder chooses a tuning factor from the factor matrix and optimizes the decoding results. The optimization improves the error performance and the improvement is shown to be iteration sensitive. This new property is a consequence of applying the tuning factor. Finally, the coding gain is calculated. The error pattern analysis shows that reducing the step size of the tuning factor, a better performance can be achieved.

6.1 Simulation Setup I

The first simulation intends to characterize the tuning factor effects by comparing the results of a range of factor values of T, assuming all other conditions set to be the same. Below, the vector **T** represents a set of tuning factor. Its elements are the coefficients to be applied to scale the extrinsic information in the iterative decoding process. The elements are chosen from the set

$$\mathbf{T} \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\} \quad (6.1)$$

6.1.1 Notations and Abbreviations

This section provides a list of notations and abbreviations used in this chapter.

d : Input bit sequence of the turbo encoder.

$\hat{d}$: Output decoded bit sequence of the turbo decoder.

v : Output codeword of the turbo encoder. (Input of the modulator).

BPSK : Binary phase shift keying modulation.

x : Output sequence of the modulator.

AWGN : Additive white Gaussian noise.

n : Additive white Gaussian noise sequence with zero-mean.

σ^2 : Variance of the AWGN.

r : Received bit sequence, corrupted by noise during transmission.

S : State sequence

SOVA : Soft output Viterbi algorithm.

MAP : Maximum a posteriori algorithm.

Log-MAP: Logarithm type of the MAP algorithm.

T : Tuning factor.

LLR : Logarithm likelihood ratio.

$L_p^{D1}, L_p^{D2'}$: Soft output a posteriori LLR from the SOVA decoders D1 and D2 respectively. Apostrophe represents the interleaved version of sequence.

L_{a1}, L_{a2}' : The a priori LLR of transmitted symbols.

L_{c1}, L_{c2}' : LLR of channel measurement.

$L_e^{D1}, L_e^{D2'}$: Extrinsic LLR from decoders D1 and D2.

6.1.2 System Model

In this section, the system model for the decoding methods to be considered will be introduced. The system block diagram is shown in Figure 6.1.

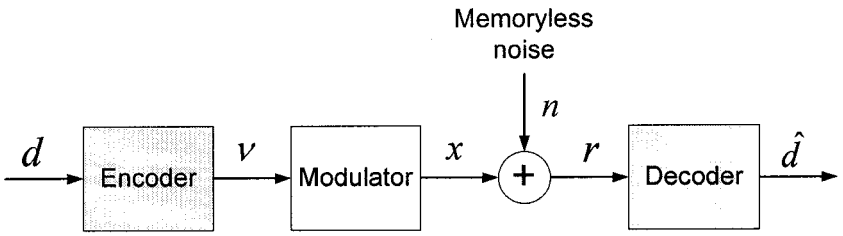


Figure 6.1: System Model

The binary message sequence fed to the input of the turbo encoder is denoted **d** and is given by

$$\mathbf{d} = (d_1, d_2, \cdots\cdots, d_t, \cdots\cdots, d_N), \tag{6.2}$$

where d_t is the message sumbol at time t and N is the sequence length. The message is encoded by a linear code. In general the message symbols d_t can be nonlinear but for simplicity we assume that they are independently generated binary symbols and have equal a priori probability. The encoding operation is modeled as discrete time

finite-state Markov process. This process can be graphically represented by state and trellis diagrams. In response to the input d_t, the finite-state Markov process generates an output v_t and changes its state from S_t to S_{t+1}, where t+1 is the next time instant. The process can be completely specificed by the following two relationships

$$v_t = f(S_t, c_t, t) \tag{6.2}$$

$$S_{t+1} = g(S_t, c_t, t) \tag{6.3}$$

The functions f( ) and g( ) are generally time varying.

The state sequence from time 0 to t is denoted by S_0^t and is written as

$$S_0^t = (S_0, S_1, \cdots\cdots, S_t) \tag{6.4}$$

The state sequence is a Markov process, so that the probability $P(S_{t+1} \mid S_0, S_1, \ldots, S_t)$ of being at S_{t+1} at time (t+1), given all states up to time t, depends only on state S_t, at time t

$$P(S_{t+1} \mid S_0, S_1, \cdots, S_t) = P(S_{t+1} \mid S_t) \tag{6.5}$$

The encoder output sequence from time 1 to t is represented as

$$v_1^t = (v_0, v_1, \cdots\cdots, v_t) \tag{6.6}$$

where

$$v_t = (v_{t,0}, v_{t,1}, \cdots\cdots, v_{t,n-1}) \tag{6.7}$$

is the code block of length n.

The code sequence v_1^t is modulated using a BPSK modulator. The modulated sequence is denoted by x_1^t and is given by

$$x_1^t = (x_0, x_1, \cdots\cdots, x_t) \tag{6.8}$$

where

$$x_t = (x_{t,0}, x_{t,1}, \cdots\cdots, x_{t,n-1}) \tag{6.9}$$

and

$$x_{t,i} = 2v_{t,i} - 1, \qquad i = 0, 1, \cdots\cdots, n-1 \tag{6.10}$$

As there is a one-to-one correspondence between the code and the modulated sequence, the encoder/modulator pair can be represented by a discrete-time finite-state Markov process and can be graphically described by state or trellis diagrams.

The modulated sequence x_1^t is corrupted by additive white Gaussian noise (AWGN), resulting in the received sequence

$$r_1^t = (r_{t,0}, r_{t,1}, \cdots\cdots, r_{t,n-1}) \tag{6.11}$$

where

$$r_t = (r_{t,0}, r_{t,1}, \cdots\cdots, r_{t,n-1}) \tag{6.12}$$

$$r_{t,i} = x_{t,i} + n_{t,i}, \qquad i = 0, 1, \cdots\cdots, n-1 \tag{6.13}$$

where $n_{t,i}$ is a zero-mean Gaussian noise random variable with variance σ^2. Each noise sample is assumed to be independent from all other samples.

The decoder produces an estimate of the input to the discrete finite-state Markov source by examining the received sequence r_1^t. The decoding problem can be alternatively formulated as finding the modulated sequence x_1^t or the coded sequence v_1^t. As there is a one-to-one correspondence between the sequences v_1^t and x_1^t, if one of them has been estimated, the other can be obtained by simple mapping.

The discrete-time finite-state Markov source model is applicable to a number of systems in communications, such as in linear convolutional and block coding, continuous phase modulation and channels with intersymbol interference.

6.1.3 Encoder Configuration

The tuning factor effect can be observed in both turbo coding and multiple turbo coding simulations. For simplicity, the simulation is conduct in the regular turbo codes scheme which has only two component encoders. The recursive systematic convolutional (RSC) encoders are symmetric and of memory order v=2. The code generator polynomial is G=[1, 5/7], rate=1/3 (un-punctured), frame size is 1024 bits per frame. Pseudo-random interleaver is used. Figure 6.2 is the turbo encoder configuration.

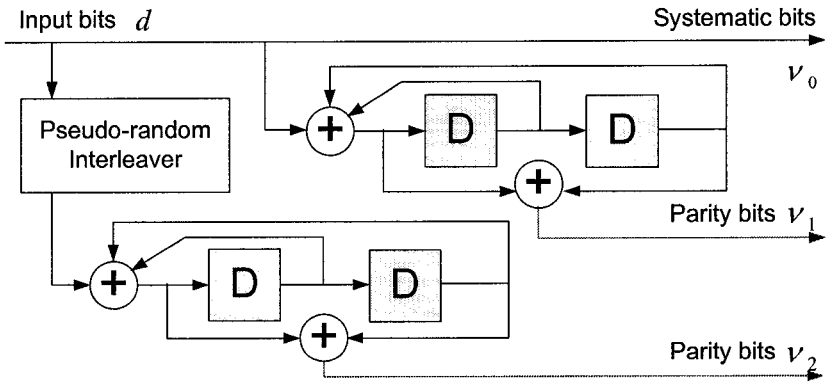


Figure 6.2: Encoder Configuration Diagram

The output of the encoder is fed to the modulator. In the system model shown in Figure 6.1, binary phase shift keying (BPSK) is used in this simulation. In general, we can consider M-ary modulation where the code sequence v is divided into groups of $\log_2 M$ binary symbols and each of them is mapped into an M-ary symbol in the modulator.

6.1.4 Decoder Configuration

In this simulation, soft output Viterbi algorithm is adopted to decode the received signal. Although the SOVA can't provide the best performance, the simulation purpose is to compare the results and examine the tuning factor effect. SOVA reduces the complexity and decoding time around 30% in respect to the Log-MAP algorithm [40].

During the iterative decoding process, the tuning factor is applied to scale the extrinsic information before feeding it to each component decoder. Figure 6.3 schematically depicts the encoder configuration.

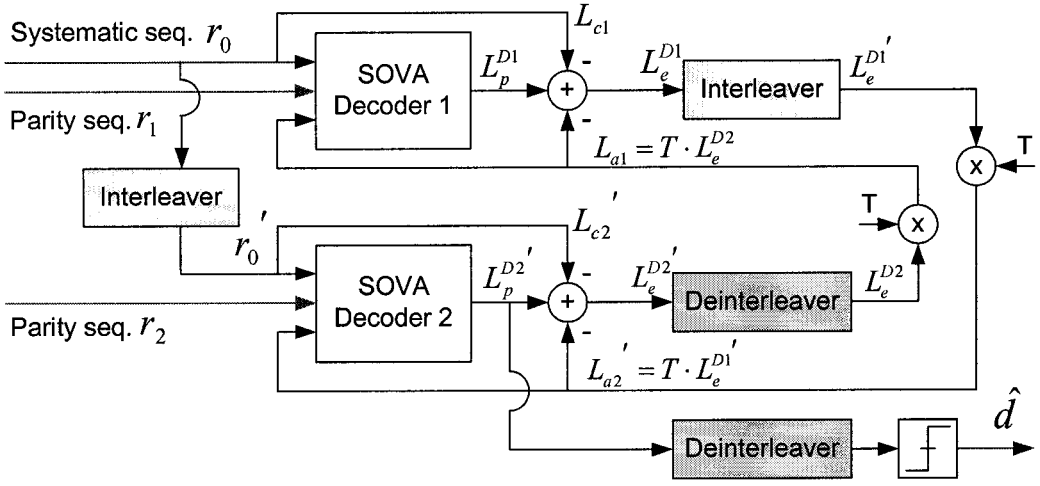


Figure 6.3: Tuning Factor Applied SOVA Decoder Configuration

The first comparison is made between T=1.00 and T=0.90. Then comparison of the results obtained from using a range of different tuning factors will be presented. Again, $\mathbf{T} \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\}$ is the set of tuning factor used.

6.2 Simulation Results I

To obtain reliable results from the Monte Carlo simulation, a certain number of frames must be transmitted and decoded then we can calculate the average error performance in terms of bit error rate (BER). The more the frames being simulated, the more accurate the results will be. A simulation can be terminated only if we collect enough errors. Intuitively, at high SNR's (E_b/N_0 > 1.5 dB in this encoder and decoder scheme), this requires numerous frames to be transmitted because the frame error rate (FER) is low at high SNR's. The stopping criteron is set to collect "50" errors in iteration 8 at low SNR and "30" errors at high SNR in the simulation.

6.2.1 Monte Carlo Simulation with Single Tuning Factor

Table 6.1 and 6.2 shows the number of frames calculated at each SNR. Table 6.3 and 6.4 is the statistics BER and FER for T=1.00 and T=0.90 at iteration 8. Figure 6.4 and 6.5 depict the corresponding error performance in BER and FER, respectively. From Figure 6.4, the coding gain is approximately 0.25 dB at both BER = $1*10^{-4}$ and $1*10^{-5}$.

Table 6.1: Number of Frames Simulated for T=1.00

E_b/N_0 (dB)	0.4	0.6	0.8	1.2	1.6	2.0	2.2	2.4	2.6
# of frames	50	61	78	285	972	3413	5972	9971	6257
# of errors	50	50	50	50	50	50	50	50	24

Table 6.2: Number of Frames Simulated for T=0.90

E_b/N_0 (dB)	0.4	0.6	0.8	1.0	1.2	1.6	1.8	2.0	2.2
# of frames	52	73	137	251	580	2238	2187	4715	5944
# of errors	50	50	50	50	50	50	30	30	25

Table 6.3: BER and FER of T=1.00 at Iteration 8

E_b/N_0 (dB)	0.4	0.6	0.8	1.2	1.6	2.0	2.2	2.4	2.6
BER	7.0e-2	3.8e-2	1.8e-2	3.5e-3	4.0e-4	5.7e-5	2.7e-5	1.7e-5	1.1e-5
FER	1.0e0	8.2e-1	6.7e-1	1.9e-1	5.1e-2	1.4e-2	8.4e-3	4.9e-3	3.8e-3

Table 6.4: BER and FER of T=0.90 at Iteration 8

E_b/N_0 (dB)	0.4	0.6	0.8	1.0	1.2	1.6	1.8	2.0	2.2
BER	4.2e-2	2.1e-2	9.1e-3	3.6e-3	9.3e-4	1.0e-4	5.4e-5	2.4e-5	1.3e-5
FER	9.9e-1	6.8e-1	3.6e-1	2.0e-1	8.6e-2	2.2e-2	1.4e-2	6.4e-3	4.2e-3

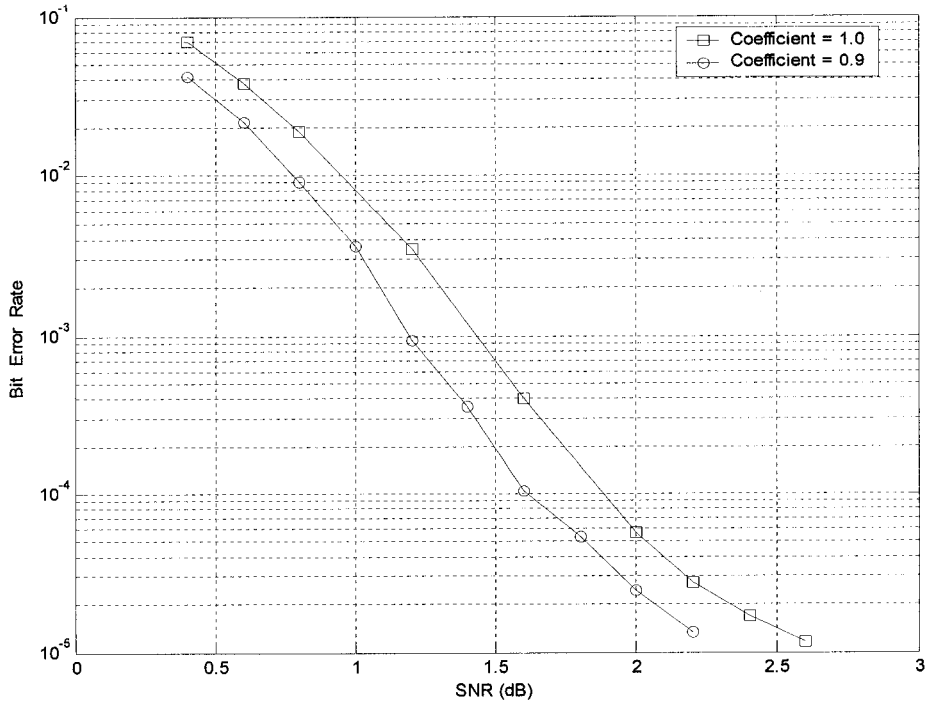


Figure 6.4: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.90

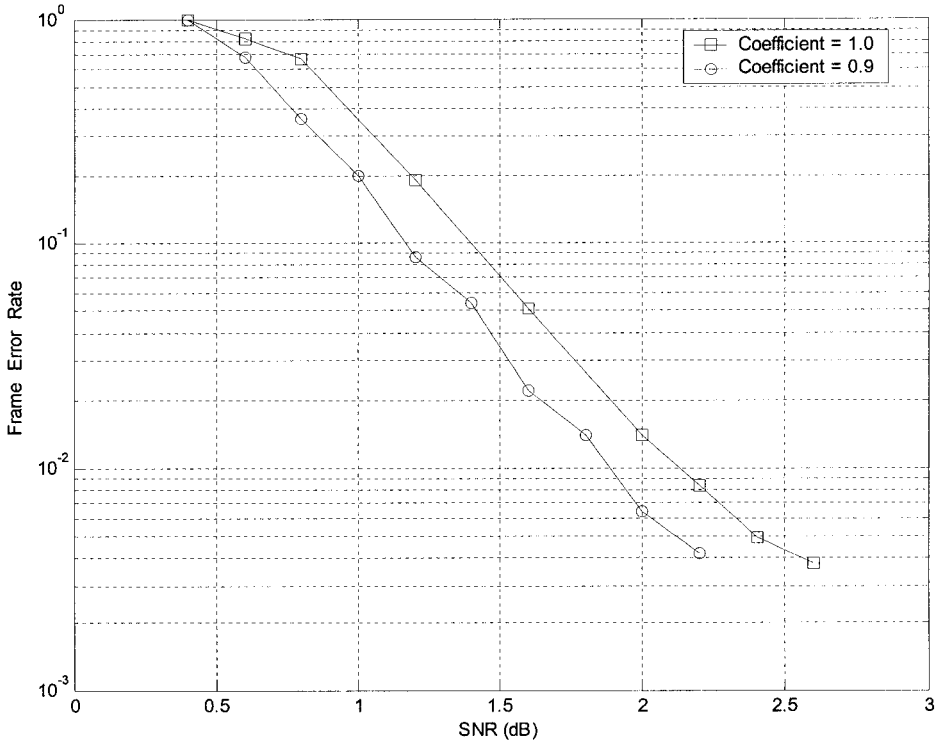


Figure 6.5: FER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.90

6.2.2 Tuning Factor Simulation Over a Range of Values

The purpose of this simulation is to compare the results when a range of tuning factors is applied in the decoding algorithm. The results are used to find the coefficient that yields the best performance. Figures 6.6 to 6.11 show the BER for tuning factor T=0.4, 0.5, 0.6, 0.7, 0.8, 1.10, respectively. In each case, the result is compared with result of the case T=1.00 (regular turbo coding). Figure 6.12 depict an overall comparison.

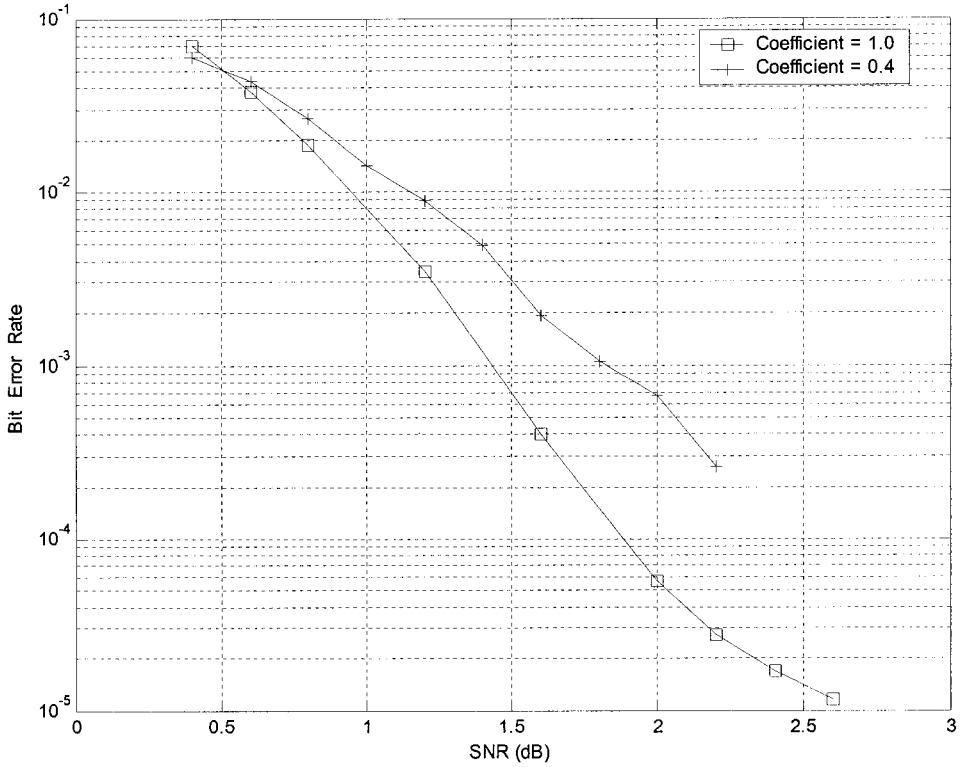


Figure 6.6: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.40

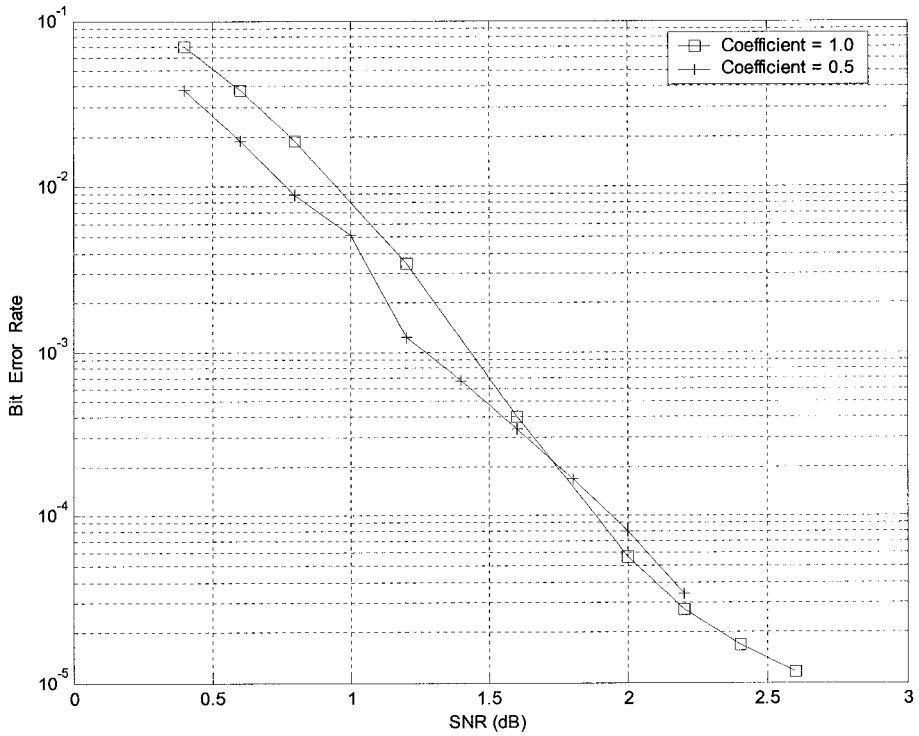


Figure 6.7: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.50

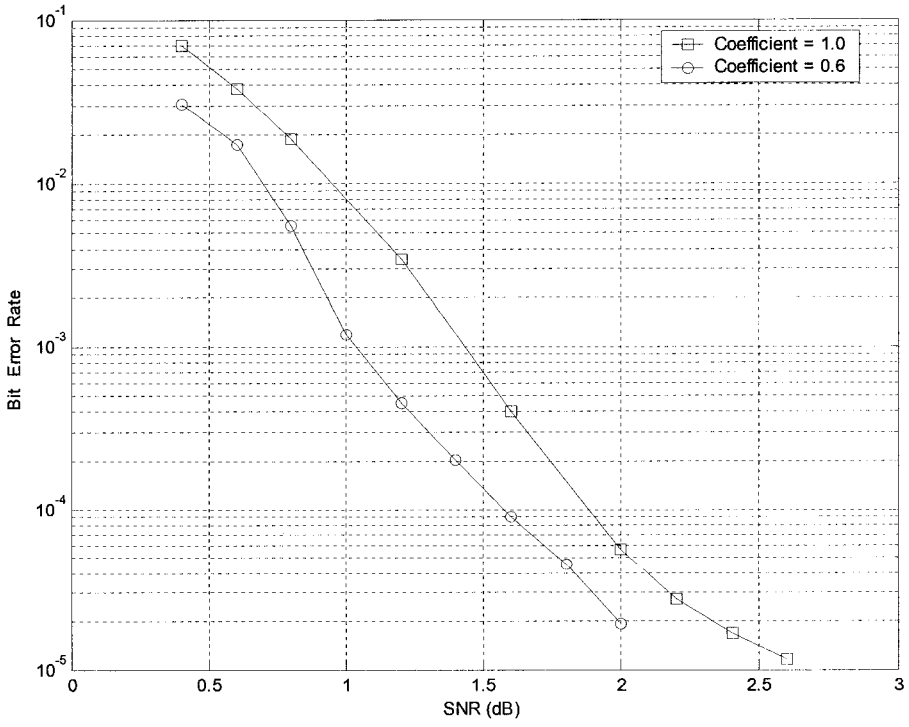


Figure 6.8: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.60

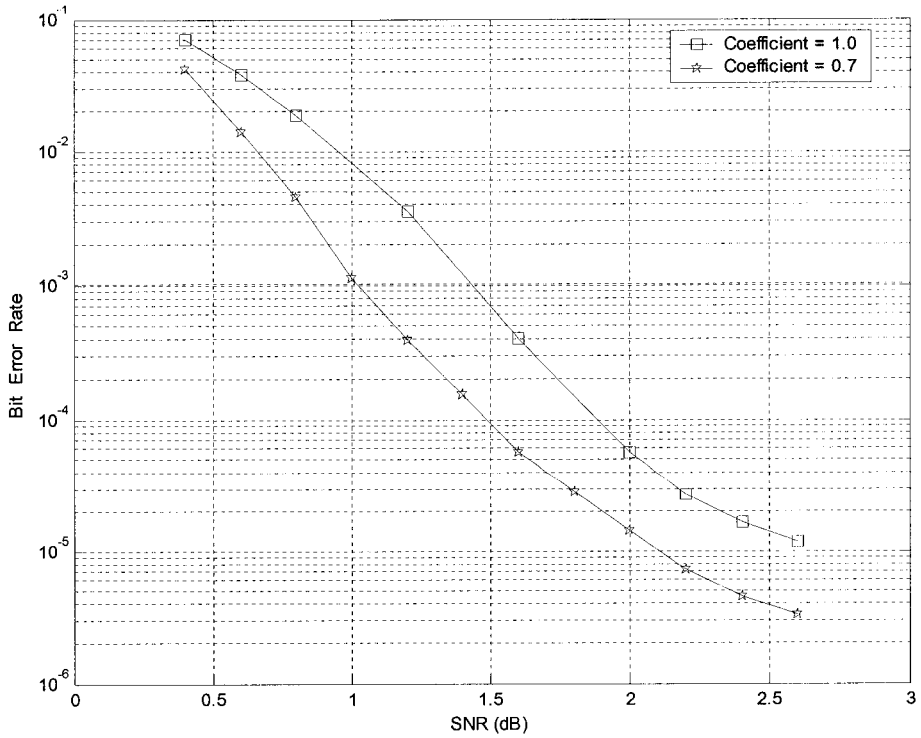


Figure 6.9: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.70

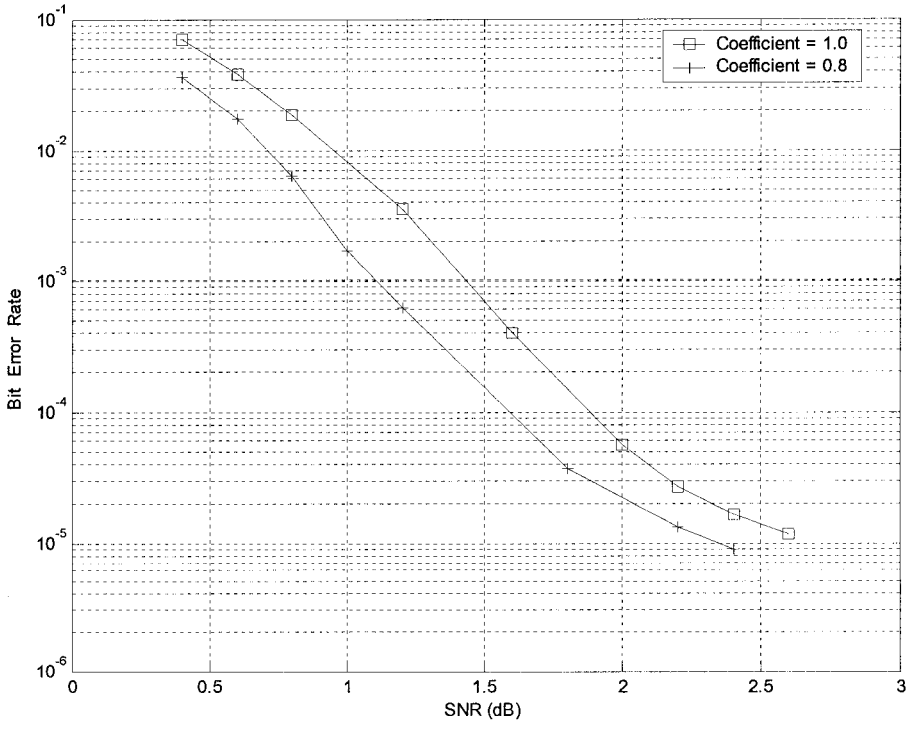


Figure 6.10: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=0.80

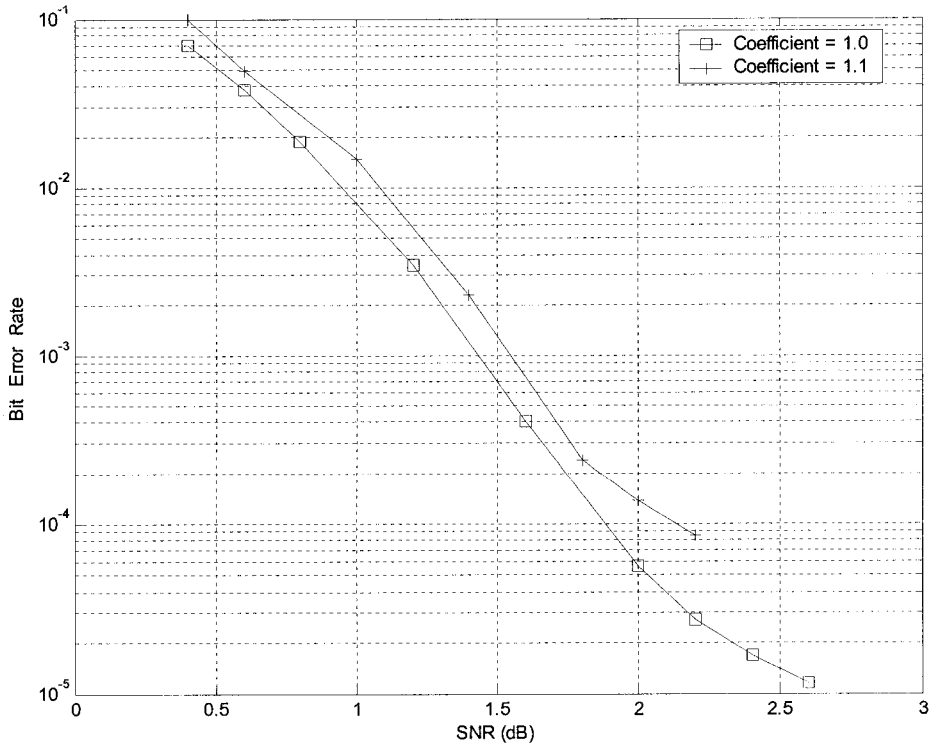


Figure 6.11: BER Comparison of Tuning Factors (Red) T=1.00 (Blue) T=1.10

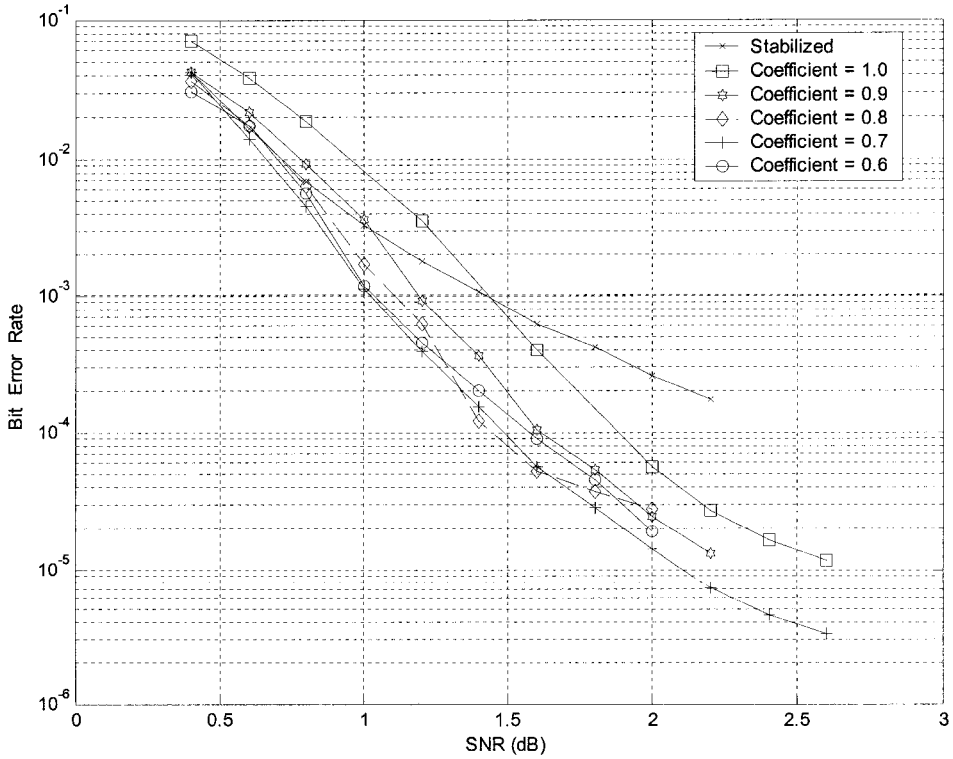


Figure 6.12: BER Comparison of T=0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1

The results show that among the tested tuning factors, T=0.70 is the coefficient which provides the best error performance in terms of BER. From Figure 6.9, we estimate that the coding gain is around 0.45dB at BER = $1*10^{-5}$.

Other than a better error performance, the error floor that occurs at high SNR can also be lowered using a tuning factor. Notice that the interleaver we use in this simulation is a pseudo-random interleaver. To further lower the error floor, an S-random interleaver or a code-matched interleaver can be applied [22, 24].

Another important behavior of a turbo code has been observed in this simulation. During the iterative decoding, some error sequences that c an't be decoded using a certain tuning factor are saved for error pattern analysis. When these error sequences are decoded by other tuning factors, surprisingly large portion of them were successfully decoded. The tuning factor acts as a correction of the overestimation of the decoding algorithm [41], as well as a correction of the mismatch between the variance of the extrinsic information and an assumption value 1. This implies that by selecting an appropriate tuning factor, it is possible to decode a highly corrupted error sequence.

6.3 Simulation Setup II

Based on the previous discussion, the idea that the factor we apply is tunable provides a guiding design principle for a new decoding scheme that can be incorporated in this simulation. With error detection techniques incorporated in the encoding process, it is possible for the receiver to know whether the received signal is successfully decoded or not, hence potentially improve decoding performance.

6.3.1 Encoder Configuration

In this simulation, an error detection code is incorporated with the turbo encoder. One of the most used error detection code is cyclic redundancy check (CRC) code. The CRC encoder can be implemented using shift registers. Figure 6.13 shows the CRC encoder structure.

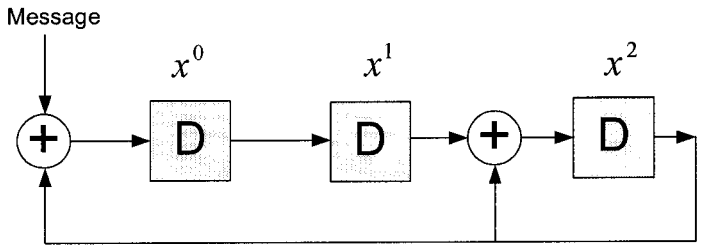


Figure 6.13: CRC Encoder Implementation using Shift Registers

To apply CRC in the turbo encoder scheme, we can serially concatenate CRC as the outer code with an inner turbo code. Figure 6.14 depicts the serial concatenation configuration.

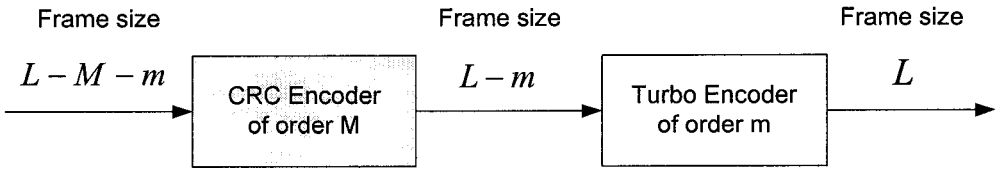


Figure 6.14: Serial Concatenation of CRC Encoder and Turbo Encoder

The input sequence to the CRC encoder is of length (*L-M-m*) if the desired turbo codeword is of length L. This is because the CRC encoder will pad *M* bits to the input sequence where *M* is the order of the CRC encoder. The padded sequence can be divided by the CRC generating polynomial. If the decoded sequence at the receiver

side can be divided by the same polynomial, then we know the decoded sequence is error-free.

The output of the CRC encoder is of length (*L-m*), where *m* is the constraint length of RSC turbo encoder. When the turbo encoder performs trellis termination, it will pad another *m* bits to the sequence. And this makes the overall output sequence of length *L*. Table 6.5 lists some common CRC polynomials [42].

Table 6.5: Common CRC Polynomials

CRC	C(x)
CRC-8	$x^8 + x^2 + x^1 + 1$
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^1 + 1$
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + 1$
CRC-16	$x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$x^{16} + x^{12} + x^5 + 1$
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$ $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

In this simulation, CRC-CCITT is adopted. The encoder structure for CRC-CCITT is shown in Figure 6.15.

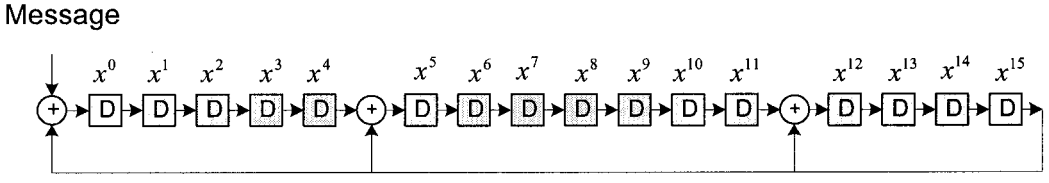


Figure 6.15: Encoder Structure of CRC-CCITT

6.3.2 Decoding Rules and Decoder Configuration

The stopping criteron of the new decoder is based on two rules. First, if the tentative decoded sequence $\hat{d}$ passes the CRC check, the decoder knows the sequence is error-free and the iterative decoding process can be stopped. This implies that a CRC checksum has to be performed at the end of each iteration. Second, if the output sequence $\hat{d}$ at the maximum number of iteration $N_{\max}$ can't pass the CRC check, then the next element in the tuning factor table will be applied to re-decode the received signal from the first iteration.

Iterative decoding can stop if the tentative output $\hat{d}$ meets the CRC check requirement; otherwise, the decoder will try all of the factors in the tuning factor table. The decoder tries to find the tuning factor which can first successfully decode the corrupted signal. If none of the tuning factor values can eliminate the errors, then the decoder stops and outputs the sequence decoded by T=0.70 based on the analytical result we get in Section 6.2.

Figure 6.16 shows the turbo decoder configuration which implements the decoding algorithm mentioned above. Since a range of tuning factors is applied in the decoding process, we call it - "Multiple Tuning Factors Turbo Decoder."

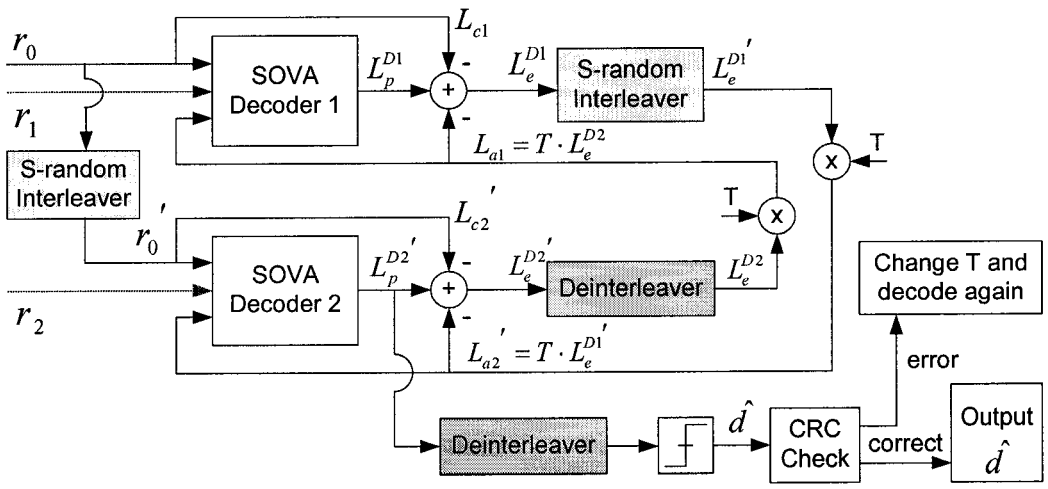


Figure 6.16: Configuration of Multiple Tuning Factors Turbo Decoder with CRC Check

Tuning factors with values smaller than 0.50 can hardly decode a highly corrupted sequence. This suggests that the elements in the tuning factor table should be in the range of 0.50 to 1.00. During the first several simulations, however, some error sequences are found to be decoded only by tuning factors with a value larger than 1.00. In the very first simulations, the largest tuning factor being applied was 1.30. Thus the elements are chosen in the range from 0.50 to 1.30.

In this section, three different simulations are performed. They are distinguished by the step size of the factors in the table. First we examine the case with a step size 0.10. Then results with step size of 0.05 and 0.01 are presented. And an overall comparison of these three cases is made.

6.4 Simulation Results II

In the following subsections, the tuning factor table of each case (step size = 0.10, 0.05, 0.01) will be given. And simulation results are shown using the standard BER

vs. SNR format. For each case, we compare the result with the error performance of a regular turbo code (T=1.00), and with the result when a single tuning factor T=0.77 is applied.

6.4.1 Multiple Factors with Step Size 0.10

We first examine the simplest case with step size 0.10. The tuning factor table of step size 0.10 is

$$\mathbf{T} = [0.70 \quad 1.00 \quad 0.80 \quad 0.60 \quad 0.90 \quad 0.50 \quad 1.10 \quad 1.20]$$

Thus **T** consists of only 8 coefficients. The order of the coefficients is base on how frequently each factor appears, but it can be rearranged. Placing frequently appearing factors in the front can speed up the decoding process since those factors are used in a first come first serve order.

To evaluate the error performance, we compare both the bit error rate (BER) and frame error rate (FER) at iteration 8 with the performance of regular turbo code with S-random interleaver, as well as the result when a single tuning factor T=0.77 is applied. The factor T=0.77 slightly outperforms T=0.70. Figure 6.17 and 6.18 present the BER and FER of the first simulation, respectively.

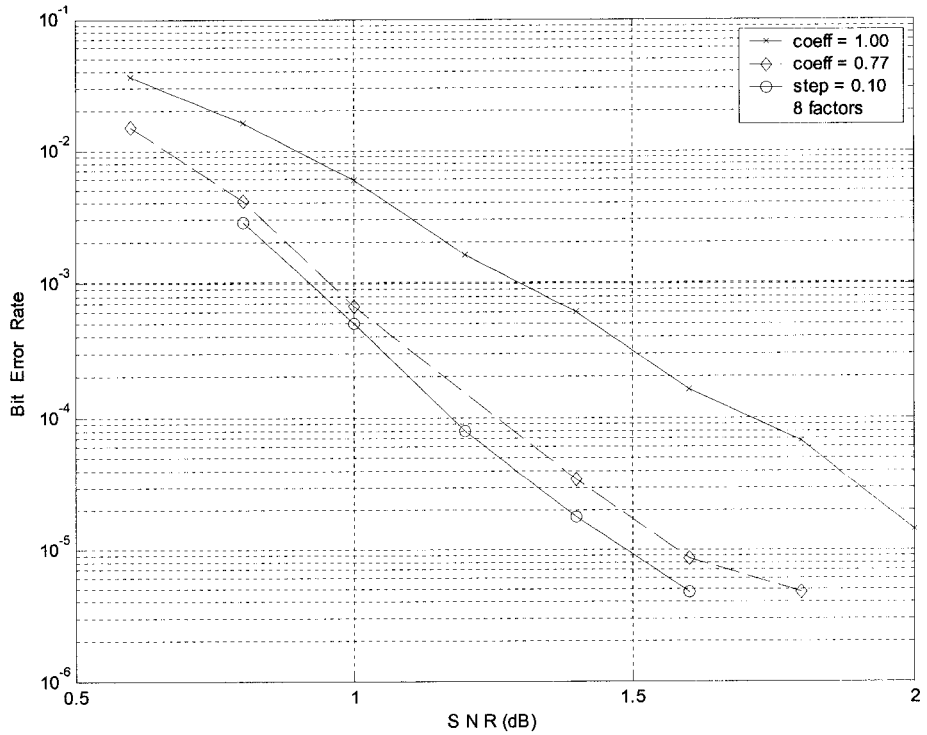


Figure 6.17: Multiple Tuning Factors Simulation Results of Step Size 0.10, BER

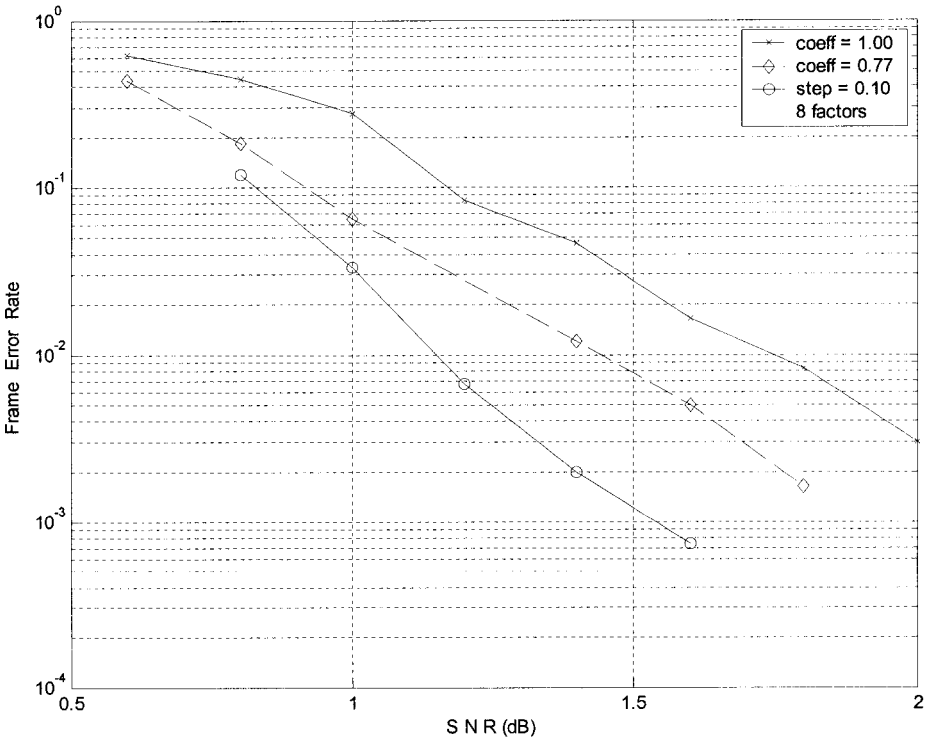


Figure 6.18: Multiple Tuning Factors Simulation Results of Step Size 0.10, FER

6.4.2 Multiple Factors with Step Size 0.05

In this case, the step size is 0.05 and the tuning factor table is

$$\mathbf{T} = \begin{bmatrix} 0.70 & 1.00 & 0.80 & 0.60 & 0.90 & 0.50 & 1.10 & 1.20 \\ 0.75 & 0.85 & 0.65 & 0.95 & 0.55 & 1.05 & 1.15 & 1.25 \end{bmatrix}$$

Sixteen factors are used in the decoding process, which is twice the number in the previous case. Simulation results are given in Figures 6.19 and 6.20 that show the BER and FER respectively. Notice that at SNR equal to 1.6 dB, the error performance is better than the result with step size 0.10. Intuitively, adding more guardian factors will lead to better results. The error performance improvement can be seen in both the BER and FER comparison.

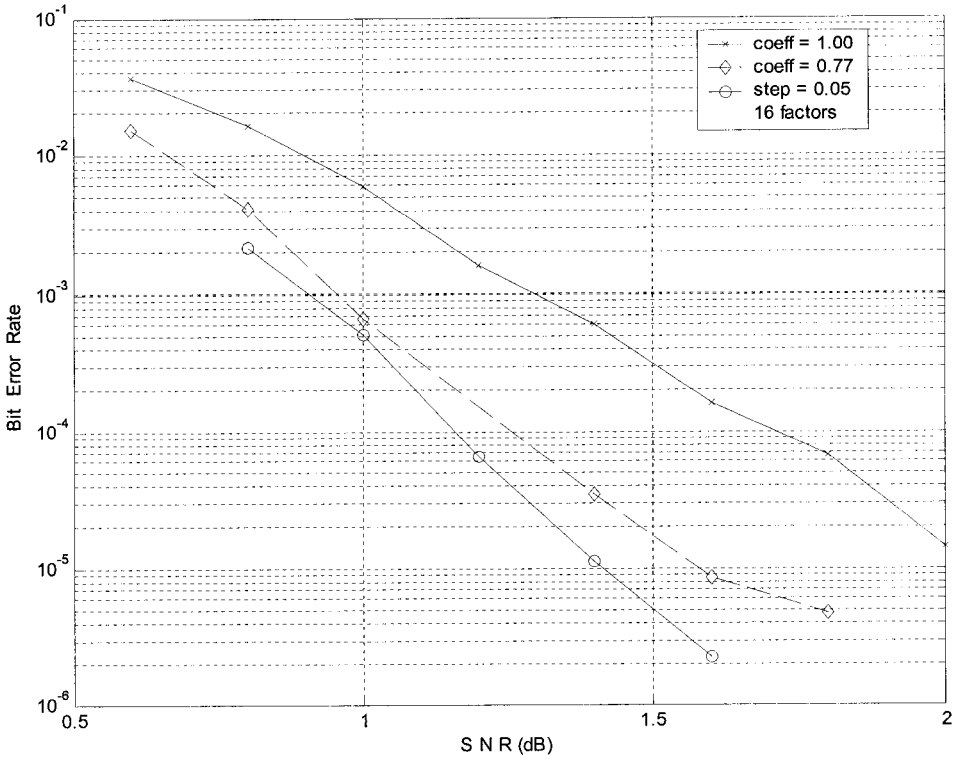


Figure 6.19: Multiple Tuning Factors Simulation Results of Step Size 0.05, BER

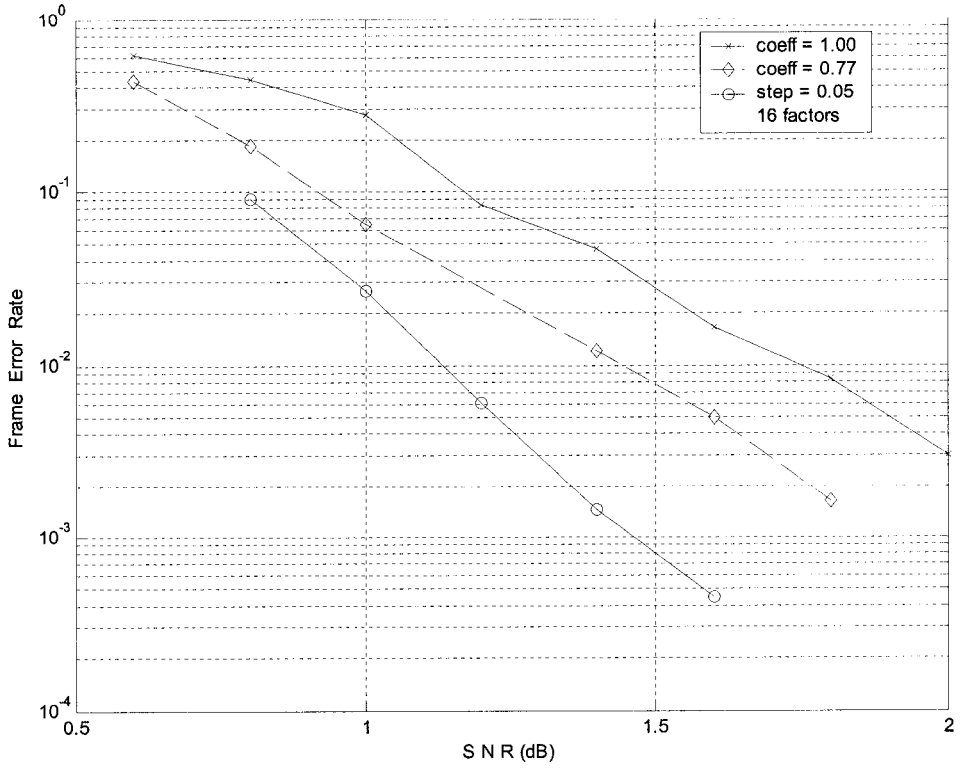


Figure 6.20: Multiple Tuning Factors Simulation Results of Step Size 0.05, FER

6.4.3 Multiple Factors with Step Size 0.01

In this case, the step size is 0.05 and the tuning factor table is

$$\mathbf{T} = \begin{bmatrix} 0.70 & 1.00 & 0.77 & 0.80 & 0.73 & 0.87 & 0.85 & 0.89 & 0.78 & 0.81 \\ 0.88 & 0.72 & 0.82 & 0.84 & 0.71 & 0.61 & 0.83 & 0.79 & 0.95 & 0.75 \\ 0.60 & 0.90 & 0.65 & 0.86 & 0.92 & 0.63 & 0.55 & 0.68 & 0.76 & 0.97 \\ 0.52 & 0.62 & 0.64 & 0.94 & 0.54 & 0.67 & 0.57 & 0.69 & 0.99 & 0.59 \\ 0.58 & 0.98 & 0.74 & 0.91 & 0.51 & 0.58 & 0.93 & 0.66 & 0.96 & 0.56 \\ 1.01 & 1.02 & 1.03 & 1.04 & 1.05 & 1.06 & 1.07 & 1.08 & 1.09 & 1.10 \\ 1.11 & 1.12 & 1.13 & 1.14 & 1.15 & 1.16 & 1.17 & 1.18 & 1.19 & 1.20 \\ 1.21 & 1.22 & 1.23 & 1.24 & 1.25 & 1.26 & 1.27 & 1.28 & 1.29 & 1.30 \quad 0.50 \end{bmatrix}$$

There are total 81 coefficients in the table. The overhead in decoding complexity is much higher than previous two cases. The BER and FER are shown in Figure 6.21 and 6.22, respectively.

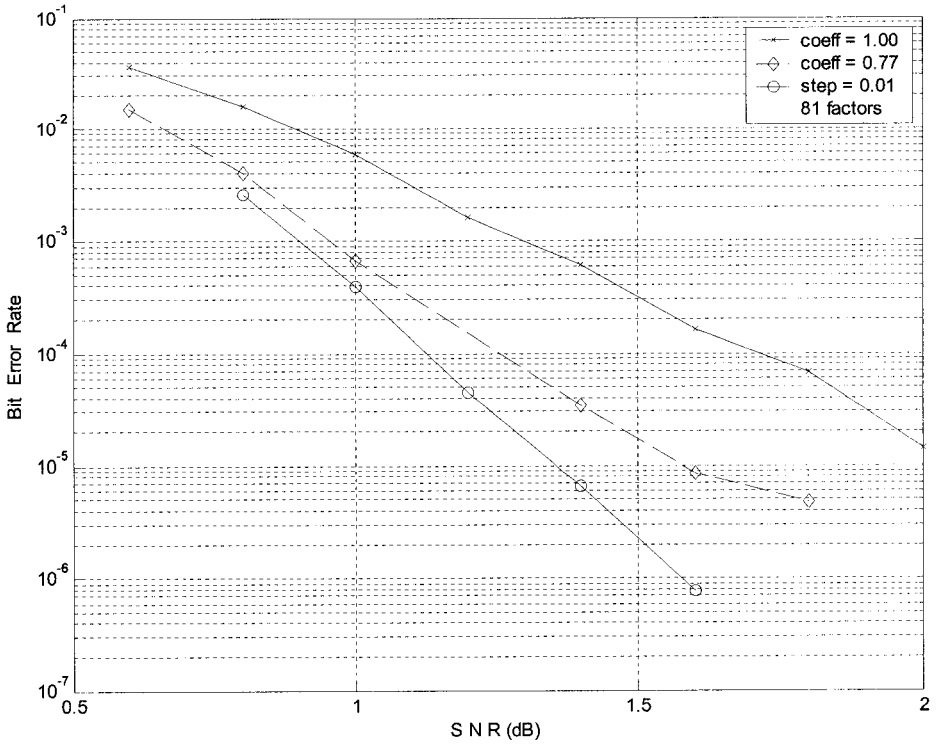


Figure 6.21: Multiple Tuning Factors Simulation Results of Step Size 0.01, BER

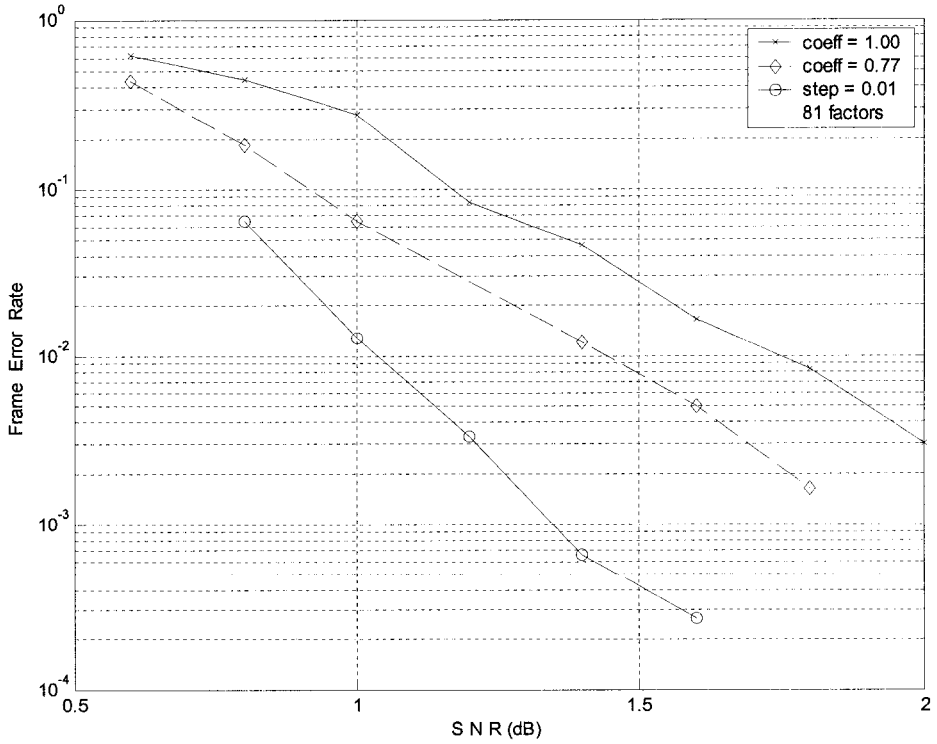


Figure 6.22: Multiple Tuning Factors Simulation Results of Step Size 0.01, FER

6.4.4 Overall Comparison

Here we compare three results obtained from simulations using tuning factors with different step size. At low SNR, the difference in error performance is not obvious. At a higher SNR, for example 1.6 dB, the improvement of using a smaller step size can be easily seen. For step size equal to 0.01, the bit error rate at 1.6 dB SNR is about $\frac{1}{10}$ its value for the single tuning factor case T=0.77. When compared with the regular turbo code, where T=1.00, the BER is nearly $\frac{1}{70}$ smaller .

The coding gain of using a single tuning factor T=0.77 at BER $1*10^{-5}$ is around 0.50 dB. For multiple tuning factors with step size 0.01, the coding gain is approximately 0.70 dB. Figures 6.23 and 6.24 present the overall BER and FER comparison of three simulation results using the multiple tuning factors decoding scheme.

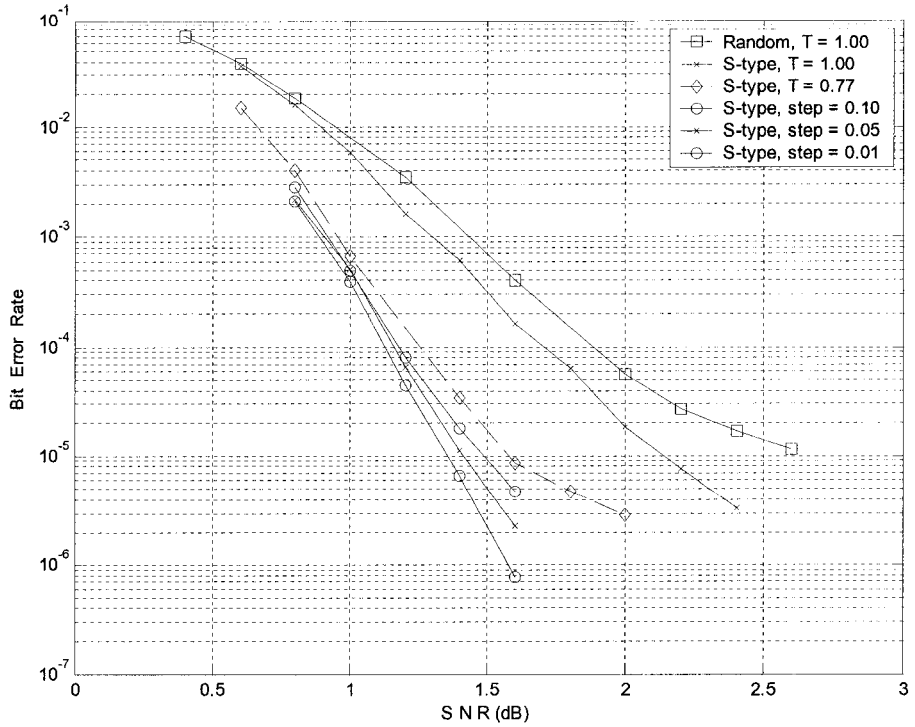


Figure 6.23: BER Comparison of Multiple Tuning Factors Simulation Results

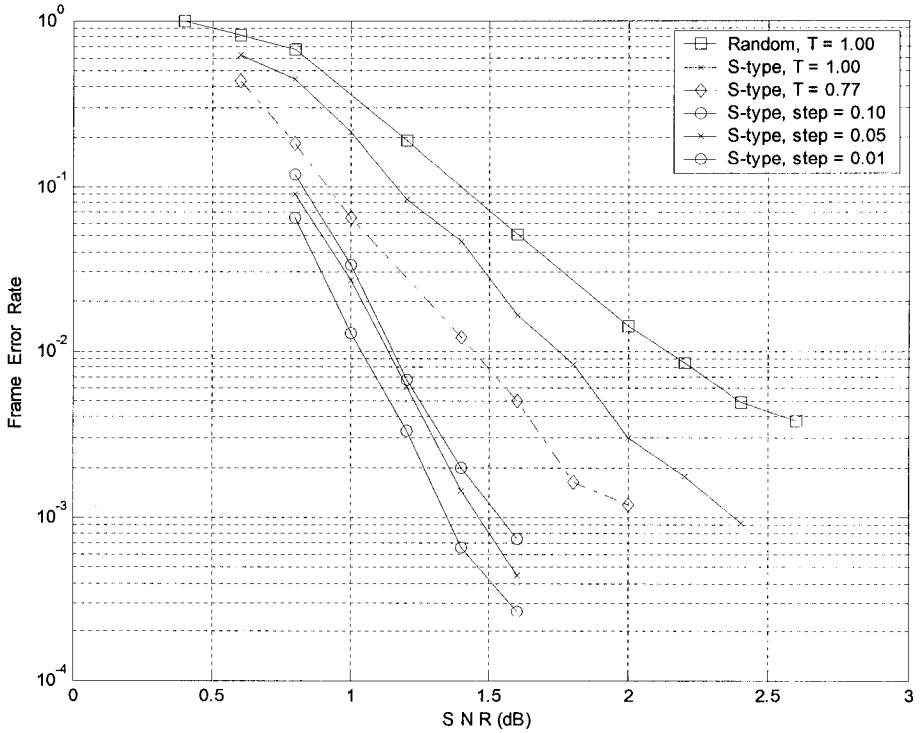


Figure 6.24: FER Comparison of Multiple Tuning Factors Simulation Results

6.5 Error Pattern Analysis

When the simulation of step size 0.01 is performed, error sequences that can't be decoded by the factors in the factor table were recorded. At SNR of 1.6 dB, 30,000 frames are transmitted and only 8 frames are in error. Among these 8 error sequences, we found that use a tuning factor with smaller step size can successfully decode some of these undecodable sequences. And some of the successfully decoded sequences can be decoded using less iterations by tuning factors with a smaller step size. Table 6.6 shows an example of decoding an error sequence by using tuning factor T=1.145, and Table 6.7 presents another example to reduce the number of iterations by applying T=1.096. The numbers in the table denote number of errors at each iteration. Again, bit error rate fluctuation can be seen in the decoding results.

Table 6.6: Example of Decoding an Error Sequence with Small Step Size Tuning Factor

# of iter	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T = 0.70	43	10	2	4	4	4	4	4	4	4	4	4	4	4	4	4
T = 1.00	43	14	4	2	2	2	2	2	2	2	4	4	4	4	4	4
T = 1.145	43	11	4	2	2	2	2	0	0	2	0	0	0	0	0	0

- The numbers in the table represents number of bit errors at the corresponding iteration. Red-colored number represents error fluctuation.

Table 6.7: Example of Reducing # of Iterations with Small Step Size Tuning Factor

# of iter	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T = 0.94	36	11	4	10	6	19	14	6	9	8	10	21	10	2	0	0
T = 1.02	36	10	4	10	11	9	6	4	18	10	3	2	2	0	0	0
T = 1.05	35	5	4	7	10	10	11	6	22	19	4	0	0	0	0	0
T = 1.10	40	7	4	9	11	9	10	20	4	2	0	0	0	0	0	0
T = 1.095	40	7	4	7	8	4	6	0	2	0	0	0	0	0	0	0
T = 1.096	40	7	4	7	8	4	6	0	0	0	0	0	0	0	0	0
T = 1.097	40	7	4	9	11	9	11	8	6	20	10	2	6	12	8	10

* The numbers in the table represents number of bit errors at the corresponding iteration. Red-colored number represents error fluctuation.

From the analytical result in Tables 6.6 and 6.7, we conclude that better error performance can be achieved when tuning factors with smaller step size are applied. However, excessive overhead will be introduced unless a smart tuning-factor searching algorithm can be found to reduce the burdensome trial-and-error decoding process.

7 Conclusions and Future Work

Turbo coding is an optimized channel coding technique that can achieve near-Shannon-limit error performance in modern communication systems. The primary goal of this thesis was to investigate their properties using numerical simulation techniques. The investigation has lead to the discovery of a "tuning factor" which can significantly impact the behavior and performance of a turbo code.

First, the bit error rate fluctuation phenomenon, usually observed under low SNR conditions, can be alleviated by applying a tuning factor that scales the extrinsic information and modifies the soft output of the decoder through iterative decoding. An alternative method to counteract the fluctuation is applying a stability factor, however, it requires more computations than the tuning factor does. In Section 5.2 and 5.4 we examined this feature in detail.

Second, the convergence speed in the iterative decoding process can be accelerated when a tuning factor is introduced. Alternatively, better error performance can be obtained for the same number of iteration even if a fixed tuning factor is used. Simulation investigation of the impact of tuning factors with different values on convergence speed and error performance suggests a tuning factor T=0.70 as a near-optimal value over the range of values $0.40 \leq T \leq 1.10$ investigated.

Third, an appropriate selected tuning factor can render an undecodable sequence become decodable provided that the appropriate factor can be found. Based on this behavior and assuming that a CRC error detection is used, a new decoding scheme that optimizes error performance was designed and implemented. Simulation results in Section 6.4 demonstrated that the new design reliably decodes highly corrupted

received signal. In addition, the results demonstrate that the error floor occurs at high SNR region can be lowered as well.

Fourth, simulation results in Chapter 6 suggest a 0.5 dB coding gain when applying a single factor T=0.70 at the expense of a negligible increased hardware complexity. The error performance can be further improved and a 0.7 dB coding gain can be obtained when multiple tuning factors are applied. However this decoding scheme requires substantial amount of computations to achieve the improved performance. The multiple tuning factors method is hard to implement in real-time digital communication system, however, it may still be useful in the case of non-real-time decoding applications such as in deep space communications.

Although this work provides some important insights into the newly discovered tuning factor effects on turbo codes performance, there are a number of aspects that can be addressed in future research. First, theoretical and mathematical justifications need to be further explored. Additional work can be done to analyze the error patterns in order to find a smarter method to search for the most appropriate tuning factors that are matched to the received sequences. Moreover, simulations assuming different environments such as Rayleigh and Rician fading channels can be conducted. Finally, hardware implementation can be performed using a special DSP processor, such as the TMS320C6x from Texas Instrument, to investigate the increase in computational complexity introduced by the tuning factor in the case of real-time communication systems.

REFERENCES

[1] Bernard Sklar, " Digital Communications, Fundamentals and Applications," 2nd edition, 2000, pp 304-325.

[2] R. W. Hamming, "Error Detecting and Error Correcting Codes," Bell Syst. Tech. J., vol. 26, 1950, pp. 147-160. http://www.engelschall.com/~sb/hamming/

[3] M. J. E. Golay, "Notes on Digital Coding," Proc. IEEE, 37, pp. 637, June 1949.

[4] G.. D. Forney, "On Decoding of BCH codes," IEEE Trans. On Information Theory, vol. 11, Oct. 1965.

[5] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," SIAM J. Appl. Math, vol. 8, pp.300-304, 1960.

[6] P. Elias, "Coding for Noisy Channels," IRE Conv. Rec., col. 3, pt 4., pp. 37-46, 1955.

[7] G.. D. Forney, "Convolutoinal Codes I: Algebraic Structure," IEEE Trans. On Information Theory, vol. IT-16, no. 6, Nov. 1970, pp. 720-738.

[8] Viterbi, A. J., "Convolutional Codes and Their Performance in Communication Systems," *IEEE Transactions on Communications Technology*, Vol. COM-19, No. 5, pp. 751-772, October 1971.

[9] C. Berrou and A. Glavieux, " Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," Proc. IEEE ICC'93, pp. 1064-1070, Geneve, Switzerland, May 1993.

[10] Shannon, C.E., "A Mathematical Theory of Communication," Bell Syst. Tech. J., vol. 27, 1948, pp. 379-423, 623-656.

[11] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

[12] Bernard Sklar, " Digital Communications, Fundamentals and Applications," 2nd edition, 2000, pp 489.

[13] S. Benedetto and G. Montorsi, "Design o f parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591-600, May 1996.

[14] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *TDA Progress Report 42-126,* Jet Propulsion Lab., August 1996.

[15] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, May 1998, pp. 909-926.

[16] J. Yuan, B. Vucetic, and W. Feng, "Combined turbo codes and interleaver design." *IEEE Trans. Commun.*, vo. 47, pp. 484-487, Apr. 1999.

[17] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, vol 42, no. 2, pp. 409-428, Mar. 1996.

[18] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," TDA Progress Report 42-123, Jet Propulsion Lab., Aug. 1995, pp.44-55.

[19] T. M. Duman and M. Salehi, "New performance bounds for turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 6, pp. 717-723, Jun. 1998.

[20] A. M. Viterbi and A. J. Viterbi, "Improved union bound on linear codes for the input-binary AWGN channel with applications to turbo-codes," in *Proc. IEEE 1998 ISIT*, MIT, MA, Aug. 1998, p. 29.

[21] I. Sason and S. Shamai, "Improved upper bounds on the performance of parallel and serial concatenated turbo codes," in *Proc. IEEE 1998 ISIT*, MIT, MA, Aug. 1998, p. 30.

[22] J. Yuan, B. Vucetic, and W. Feng, "A Code-matched Interleaver Design for Turbo Codes," *IEEE Trans. on Comm., vol. 50, No. 6, June 2002.*

[23] J. R. Hess, "Implementation of a turbo decoder on a configurable computing platform," Master Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, Sept. 1999.

[24] D. Divsalar, F. Pollara, "Turbo codes for PCS applications," *Proc., IEEE Int. Conf. on Commun.*, pp.54-59, May 1995.

[25] I. Richer, "A sample interleaver for use with Viterbi decoding," IEEE Trans. Commun., vol. 26, pp. 406-408, Mar. 1978.

[26] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 477-480, May 1996.

[27] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *Proc. 1999 Int. Conf. Commun.*, vol. 2, pp. 1009-1013.

[28] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429-445, Mar. 1996.

[29] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284-287, Mar. 1974.

[30] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260-264, Feb. 1998.

[31] S. S. Pietrobon, " Implementation and performance of a serial MAP decoder for use in an iterative turbo decoder," *IEEE Int. Symp. Inform. Theory*, p. 471, Sep. 1995.

[32] Viterbi, A. J., "Convolutional Codes and Their Performance in Communication Systems," *IEEE Transactions on Communications Technology*, Vol. COM-19, No. 5, pp. 751-772, October 1971.

[33] G. D. Forney, "The Viterbi algorithm," *Proc., IEEE*, vol. 61, no. 3, pp. 268-278, Mar. 1973.

[34] J. Hagenauer and L. Papke, "Decoding "turbo"-codes with the soft output Viterbi algorithm," *Proc. of Int. Symp. on Information Theory,* p. 164, 1994.

[35] R. Y. Shao, S. Lin and M. P. C. Fossorier, "Two simple stopping criteria for turbo decoding," IEEE Trans. On Commun., vol. 47, no.8, pp. 1117-1120, Aug. 1999.

[36] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders," TMO Progress Report 42-142, Jet Propulsion Lab., Aug. 2000.

[37] Consultative Committee for Space Data Systems, "Telemetry Channel Coding," Blue Book, vol. 101.0-B-4, Issue 4, May 1999.

[38] D. Divsalar and F. Pollara, "Multiple Turbo Codes for Deep-Space Communications," TDA Progress Report 42-121, Jet Propulsion Lab., pp. 66-77, May 15, 1995.

[39] D. Divsalar and F. Pollara, "Multiple Turbo Codes," in Proc. IEEE Military Communications Conference, San Diego, CA, pp. 279-285, Nov. 1995.

[40] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *Proc. 1999 Int. Conf. Commun.*, vol. 2, pp. 1009-1013.

[41] G. Colavolpe, G. Ferrari, and R. Raheli, "Extrinsic Information in Iterative Decoding:A Unified View," IEEE Trans. On Commun., vol. 49, no. 12, pp. 2088-2094, Dec. 2001.

[42] L. L. Peterson and B. S. Davie, "Computer Networks, A System Approach," 2nd Edition, Morgan Kaufmann Publishers, 2000, pp. 96-102.